Estructuras de datos
Clase teórica 4

UNIVERSIDAD NACIONAL DE COLOMBIA SEDE MEDELLÍN

Contenido

- Colas
- Pilas

Material elaborado por: Julián Moreno

Facultad de Minas, Departamento de Ciencias de la Computación y la Decisión

Como vamos hasta el momento

TALLERES — Estructuras de datos

ESTRUCTURAS DE DATOS / Talleres / Arreglos, listas, pilas y colas / Ejercicios

Nombre	Tipo de entrada	# estudiantes que lo han resuelto	Estadisticas
K-esimos elementos	Código	50	
Suma de posiciones	Código	45	
Estan o no estan	Código	45	
Distancias entre postes	Código	35	
Inventario dinamico	Código	26	
Despidos masivos	Código	16	
Teclado numerico	Código	0	

Primer trabajo práctico

Han visto esas imágenes de "expectativa versus realidad"? He aquí una pequeña recopilación …

Para los hombres: "Espere un momento que ya viene la enfermera
Para las mujeres: "Vayamos al gimnasio a ver que pisteamos"
El 31 de octubre: "Iré a la fiesta de disfraces"

Al comenzar el semestre: "Metámonos a natación"

Para los fans de *Game of thrones:* "Me dejaré crecer la barba"

Expectativa **Realidad**

¿Qué es eso de colas y pilas?

Expectativa **Realidad**

"Trabajemos con pilas"

"Trabajemos con colas"

Colas y pilas

Las colas y las pilas son estructuras de datos ampliamente utilizadas en computación. Por ejemplo la gestión del procesador se hace en términos generales mediante una cola, mientras que la gestión de la memoria se hace mediante una pila.

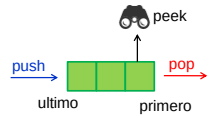
Dependiendo de la forma como se implementen tendrán más o menos métodos, pero sus tres principales son:

- push: Inserta un nuevo elemento
- pop: Extrae el elemento próximo a salir
- peek: "Espía" el elemento próximo a salir

En ambos casos el método push inserta el nuevo elemento después del último que se haya insertado.

Colas y pilas

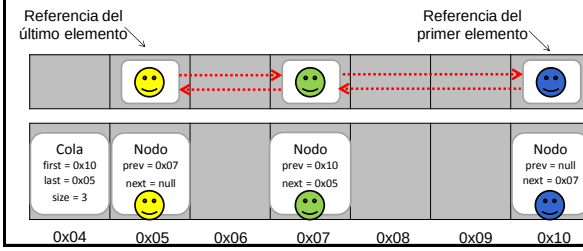
En la cola, al extraer se obtiene el elemento que se haya insertado hace más tiempo. Por esta razón también se conocen como estructuras de datos FIFO (First In First Out: primero en entrar, primero en salir).



En la pila, al extraer se obtiene el último elemento que acaba de insertarse. Por esta razón también se conocen como estructuras de datos LIFO (Last In First Out: último en entrar, primero en salir).

Implementación de una cola

Las operaciones que discutimos previamente (push, pop, peek) determinan qué debe hacer la estructura, más no el cómo. De lo que hemos visto, una cola se podría implementar tanto mediante un arreglo dinámico, como con una lista enlazada, sin embargo, en procura de la eficiencia, lo común es la segunda alternativa con doble enlace y con dos referencias, una para el primero, y otra para el último elemento.



Inserción en una una cola

Comencemos analizando la operación de inserción. Si nos referimos al *push*, dicha inserción (*push*) se realiza al final de la lista lo cual, como ya vimos, tiene una eficiencia *O(1)*:

```
// e es el elemento a ingresar
p = new nodo
p.elm = e
if (cola.size > 0){
   (cola.last).next = p
   p.prev = cola.last
}
else
   cola.first = p
cola.last = p
cola.size++
```

Borrado en una cola

Analicemos ahora la operación de borrado. Si nos referimos al *pop*, dicho borrado + devolución se realiza al inicio de la lista lo cual, como ya vimos, tiene una eficiencia *O(1)*:

```
if (cola.size > 0){
   p = (cola.last).elm
   cola.first = (cola.first).next
   cola.size–
   return p
}
else
   return ERROR
```

Indexado en una cola

Si lo analizamos, hacer un peek corresponde a hacer un indexado sobre el índice 0 de la lista lo cual, como ya vimos, tiene una eficiencia *O(1)*:

```
if (cola.size > 0){
   p = (cola.last).elm
   return p
}
else
   return ERROR
```

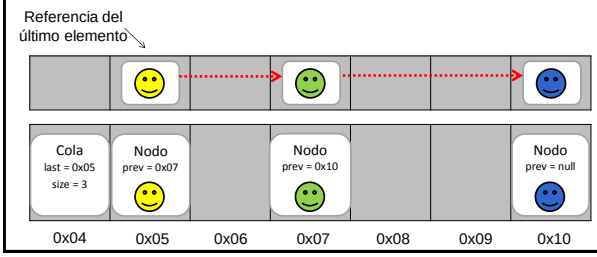
Búsqueda en una cola

La búsqueda en una cola no suele ser una operación común. Sin embargo, en caso de ser necesaria, esta operación resulta ser casi idéntica que en una lista enlazada (como es doblemente enlazada se puede hacer tanto desde el primero hacia el último como desde el último hacia el primero) y por tanto tiene la misma eficiencia *O(n)*:

```
// e es el elemento buscado
p = cola.first
i = 0
while (i < cola.size && p.elm != e){
   p = p.next
   i++
}
if (p.elm = e)
   return i
else
   return -1
```

Implementación de una pila

Una pila se puede implementar casi igual que una cola. Sin embargo, considerando que tanto la inserción (*push*) como el borrado (*pop*) se realizan por el mismo extremo de la lista, solo necesitamos una referencia al elemento de ese extremo y un enlace simple entre los nodos:



Inserción en una pila

La inserción (*push*) se realiza al final de la lista lo cual, al igual que en la cola, tiene una eficiencia *O(1)*, aunque tener un solo enlace lo hace de hecho es más simple:

```
// e es el elemento a ingresar
p = new nodo
p.elm = e
if (pila.size > 0)
   p.prev = pila.last
pila.last = p
pila.size++
```

Borrado en una pila

Igual que en la cola, esta operación (refiriéndonos al *pop* = borrado + devolución) se puede hacer en *O(1)*:

```
if (pila.size > 0){
   p = (pila.last).elm
   pila.last = (pila.last).prev
   pila.size–
   return p
}
else
   return ERROR
```

Indexado en una pila

Al igual que en las colas, el indexado que nos interesa es el peek, el cual de nuevo tiene una eficiencia *O(1)*:

```
if (pila.size > 0){
   p = (pila.last).elm
   return p
}
else
   return ERROR
```

Búsqueda en una pila

Resulta similar al de una cola solo que la búsqueda lineal solo se puede realizar en un sentido: desde el último hacia el primero, pero al fin y al cabo con la misma eficiencia *O(n)*:

```
// e es el elemento buscado
p = pila.last
i = 0
while (i < pila.size && p.elm != e){
   p = p.prev
   i++
}
if (p.elm = e)
   return i
else
   return -1
```

Ejercicio

Para poner a prueba parte de lo aprendido, hagamos una prueba de escritorio al siguiente pseudocódigo, ¿qué mostraría?

```
cola c
c.push(10)
c.push(25)
c.push(18)
print(c.pop())
c.push(34)
print(c.peek())
c.push(27)
print(c.pop())
print(c.peek())
print(c.pop())
```

Ejercicio

Hagamos algo parecido pero ahora con una pila, ¿qué mostraría?

```
pila p
p.push(10)
p.push(25)
p.push(18)
print(p.pop())
p.push(34)
print(p.peek())
p.push(27)
print(p.pop())
print(p.peek())
print(p.pop())
```

Ejercicio

Finalmente, miremos este pseudocódigo que emplea ambas estructuras, ¿qué mostraría?

```
cola c
pila p
c.push(50)
c.push(70)
p.push(60)
p.push(40)
print(c.peek())
print(p.pop())
p.push(c.pop())
c.push(p.peek()-10)
print(p.peek())
print(c.pop())
```

Tabla resumen

Recapitulando la clase de hoy tenemos que:

Estructura	Inserción	Indexación	Búsqueda	Borrado
Cola	*O(1)* refiriéndonos al push	*O(1)* refiriéndonos al peek	*O(n)*	*O(1)* refiriéndonos al pop
Pila	*O(1)* refiriéndonos al push	*O(1)* refiriéndonos al peek	*O(n)*	*O(1)* refiriéndonos al pop