# Algoritmos y estructuras de datos

## Pilas y colas en listas enlazadas

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas



23 de agosto de 2024

Una pila $s$ tiene dos operaciones:

1. apila$(s, x)$ agrega el dato $x$ a la pila $s$ y
2. desapila$(s)$ regresa el último dato agregado a $s$ y lo elimina.

Adicionalmente queremos crear y destruir una pila y saber si está vacía.

**Listas enlazadas**

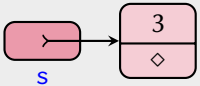
Si lo pensamos bien, prácticamente todas estas operaciones son idénticas a las que ya conocemos para una lista enlazada: una pila será una lista enlazada en la que siempre se insertan o eliminan nodos al principio.

# Tipo de datos abstracto pila

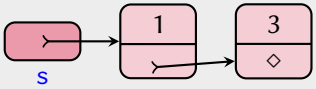
## Ejemplo de operaciones de pila en una lista enlazada

Pila vacía `s = NULL;`

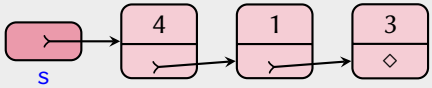

`apilaLE(&s, 3);`



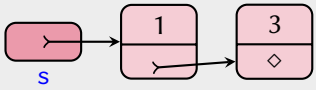

`apilaLE(&s, 1);`



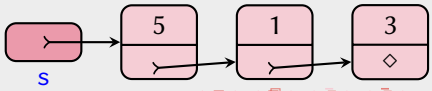

`apilaLE(&s, 4);`




`desapilaLE(&s, &x);`




`apilaLE(&s, 5);`



# Pila en una lista enlazada

## Estructura de datos

Definiremos un tipo estructurado `nodo` para representar un nodo y también un tipo `pilaLE` para representar una pila como una lista enlazada. El tipo `nodo` consiste de un dato `a` y un apuntador `sig` al siguiente nodo (que valdrá `NULL` si no hay siguiente nodo).

```
typedef struct nodo {
  int a;             // dato almacenado
  struct nodo *sig; // enlace al siguiente
} nodo;
```

Por otro lado, el tipo `pilaLE` es un apuntador a `nodo`.

```
typedef nodo *pilaLE;
```

Note que los tipos `nodo *`, `struct nodo *` y `pilaLE` son equivalentes.

# Pila en una lista enlazada

## Pilas vacías

Declaramos una pila vacía `s` con `pilaLE s = NULL;` De la misma manera, sabremos que la pila está vacía si `s == NULL`.


Destruir una pila es lo mismo que destruir una lista enlazada.

```
void destruyePilaLE(pilaLE *s) {
  int x;
  while (desapilaLE(s, &x));
}
```

Esto es lo mismo que insertar un nodo al inicio de una pila.

```
void apilaLE(pilaLE *s, int x) {
  insertaNodo(s, x);
}
```

Esto es lo mismo que eliminar un nodo del principio de la pila.

```
int desapilaLE(pilaLE *s, int *x) {
  return eliminaNodo(s, x);
}
```

# Tres representaciones de pilas

## Resumen de resultados

Cantidad de operaciones en el peor de los casos, actuando sobre una pila $s$ de hasta $n$ elementos y un elemento $x$.

| Operación | Arreglo estático | Arreglo dinámico | Lista enlazada |
|---|---|---|---|
| crear | 1 | 1 | 1 |
| destruir | 1 | 1 | $n$ |
| vacía | 1 | 1 | 1 |
| apilar | 1 | $n+1$ | 1 |
| desapilar | 1 | 1 | 1 |

Recordemos que la $n+1$ en la pila en arreglo dinámico es en promedio una constante menor que 3. La $n$ en la lista enlazada sólo importa si no se usa todo el contenido de la pila: sólo se hace una vez y es en promedio una constante.

Una cola $s$ tiene dos operaciones:

1. forma$(s, x)$ agrega el dato $x$ a la cola $s$ y
2. desforma$(s)$ regresa el primer dato agregado a $s$ y lo elimina.

Adicionalmente queremos crear y destruir una cola y saber si está vacía.

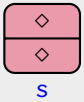
### Listas enlazadas

De las operaciones forma y desforma, una deberá actuar sobre el primer nodo de la lista enlazada y la otra sobre el último. Como vimos con la pila, es fácil hacer cualquiera de estas dos operaciones en el primer nodo. Pero hacer desforma sobre el último nodo implica saber dónde está el penúltimo, lo cual es lento. Haremos forma en el último nodo y desforma en el primero.

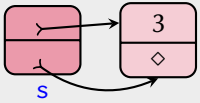
# Tipo de datos abstracto cola

## Ejemplo de operaciones de cola en una lista enlazada

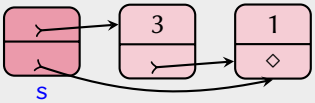
Cola vacía



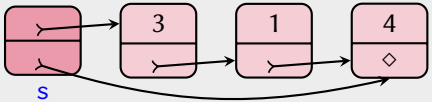

formaLE(&s, 3);



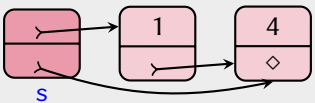

formaLE(&s, 1);



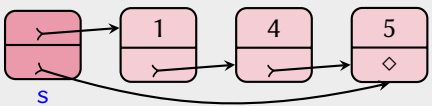

formaLE(&s, 4);




desformaLE(&s, &x);




formaLE(&s, 5);



# Cola en una lista enlazada

## Estructura de datos

Definiremos un tipo estructurado `colaLE` para representar una cola como una lista enlazada. El tipo `colaLE` consiste de dos apuntadores, uno al primer nodo y otro al último, que serán nulos si la cola está vacía y apuntarán al mismo nodo si la cola tiene exactamente un nodo.

```
typedef struct {
  nodo *ante; // al primer nodo
  nodo *tras; // al ultimo nodo
} colaLE;
```

Crear una cola vacía:

```
void creaColaLE(colaLE *s) {
  s->ante = NULL;
  s->tras = NULL;
}
```

Destruir una cola es casi lo mismo que destruir una lista enlazada:

```
void destruyeColaLE(colaLE *s) {
  int x;
  while (desformaLE(s, &x));
}
```

Esto es insertar un nodo al final de una cola. Si es el único nodo, también es el primer nodo. Si no, es el siguiente del anterior último nodo.

```
void formaLE(colaLE *s, int x) {
  nodo *t = creaNodo(x, NULL); // crea ultimo nodo
  if (s->tras == NULL)         // es el primer nodo
    s->ante = t;               // actualizamos ante
  else
    s->tras->sig = t;          // actualizamos enlace
  s->tras = t;                 // actualizamos tras
}
```

Esto es casi lo mismo que eliminar un nodo del principio de la cola. Si era el único nodo, entonces tampoco hay último nodo.

```
int desformaLE(colaLE *s, int *x) {
  if (!eliminaNodo(&(s->ante), x)) // si no hay nodo
    return 0;                      // no se pudo
  if (s->ante == NULL)             // si era el ultimo
    s->tras = NULL;                // actualizamos tras
  return 1;                        // si se pudo
}
```

# Tres representaciones de colas

## Resumen de resultados

Cantidad de operaciones en el peor de los casos, actuando sobre una cola $s$ de hasta $n$ elementos y un elemento $x$.

| Operación | Arreglo estático | Arreglo dinámico | Lista enlazada |
|---|---|---|---|
| crear | 1 | 1 | 1 |
| destruir | 1 | 1 | $n$ |
| vacía | 1 | 1 | 1 |
| formar | 1 | $\frac{3}{2}n + 1$ | 1 |
| desformar | 1 | $n + 1$ | 1 |

Los $n + 1$ y $\frac{3}{2}n + 1$ en la cola en arreglo dinámico son en promedio constantes. La $n$ en la lista enlazada sólo importa si no se usa todo el contenido de la cola: sólo se hace una vez y es en promedio constante.