Giovanni Urdaneta
CI: 28288477
Periodo 2022B
“Estructuras dinámicas de datos”

Colas

Son estructuras de datos que funcionan según el principio FIFO (first in, first out = el primero que entra es el primero que sale).

Podemos asimilar una cola en programación con la cola que haríamos para comprar las entradas para ir al cine o para pedir en un lugar de comida rápida; si entramos de primeros, seremos los primeros en salir de la cola. Este funcionamiento limita la cantidad de acciones que podemos realizar con dicha estructura de datos. De hecho, a la hora de manipular datos, solo contamos con 2 métodos: encolar y desencolar.

Encolar consiste en agregar un elemento, y solo puede agregarse desde el final de la cola. Desencolar consiste en eliminar o extraer un elemento, y tiene que ser el elemento del principio. Es decir, el proceso de encolar es el mismo que sucede cuando entramos en la cola para comprar la entrada del cine, y el proceso de desencolar es el que se ejecuta cuando ya compramos la entrada y estamos fuera de la cola.

Esto implica que no podemos “indexar” una cola, al igual que lo haríamos con un array. Es decir, no podemos agregar un elemento en cierta ubicación o eliminarlo. Es una estructura totalmente estática de la cual, por funcionamiento, no podemos alterar su orden.

Las colas nos sirven para controlar procesos, para automatizar por orden de llegada los hilos o sucesos que debemos procesar, bajo cualquier escenario. Es una estructura de datos que se adapta a las circunstancias. Por ejemplo: las playlist de cualquier reproductor de música utilizan colas como estructuras de datos.

Listas

Son, al igual que los arrays, estructuras secuenciales de datos, pero esta tiene una diferencia primordial con la anterior: cada elemento de la lista se divide en 2.

En una lista (también denominada: lista enlazada), cada elemento tiene un determinado valor (que puede ser de tipo primitivo o no primitivo) y tiene un puntero que apunta hacia el siguiente elemento de la lista. Esto implica que el puntero del último elemento apunta hacia NULL. Esto quiere decir que, en los lenguajes altamente tipados, no es necesario reservar la cantidad de elementos que tendrá una lista, como si es necesario hacerlo con arrays.

Los métodos primordiales de una lista son prácticamente iguales a los que utilizamos con los arrays. Podemos insertar y eliminar elementos desde cualquier lugar de la estructura.

La utilidad que tiene esta estructura es el ahorro de memoria; a la hora de iterar sobre las listas no se gasta tanta memoria como la que se utiliza en los arrays, lo que las hace más eficientes, a pesar de ser más complejas de programar.

Cada elemento de una lista normalmente se denomina “nodo”.

Implementación en C

Ambas estructuras están interconectadas, debido a que debemos hacer uso de las listas enlazadas para poder implementar las colas de una manera eficiente. Ciertamente podríamos hacerlo con arrays, pero sería inutil, debido a que:

1. Sería susceptible a que su orden sea alterado.
2. Se gastaría más memoria.
3. En el caso de un lenguaje de tipado alto, como lo es C, tendríamos que reservar el número de elementos. Y esta mecánica no es eficaz para cumplir con el funcionamiento de las colas.

Cada elemento de la cola es un nodo, y la implementación luce más o menos así:

```c
typedef struct nodo {
    int dato;
    struct nodo *siguiente;
} nodo;

nodo *cabeza = NULL;
nodo *final = NULL;

nodo *crear_nodo(int valor){
    nodo *elemento = (nodo*)malloc(sizeof(nodo));
    elemento -> dato = valor;
    elemento -> siguiente = NULL;
    return elemento;
}

int vacia()
{
    if (cabeza == NULL){
        return 1;
    }
    else{
        return 0;
    }
}
```

```
void encolar(int valor){
    nodo *elemento = crear_nodo(valor);

    if (vacia() == 1){
        cabeza = elemento;
        final = elemento;
    }
    else{
        final -> siguiente = elemento;
        final = elemento;
    }
}
```

```
int desencolar()
{
    if (!vacia())
    {
        int informacion = cabeza -> dato;
        nodo *aux = cabeza;
        if (cabeza == final)
        {
            cabeza = NULL;
            final = NULL;
        }
        else
        {
            cabeza = cabeza -> siguiente;
        }
        free(aux);

        return informacion;
    }
```