JAVASCRIPT ALGORITMOS DE ORDENACIÓN

Un diagrama de flujo es una forma de representar los pasos de un algoritmo, a través de símbolos bien definidos para cada tipo de proceso (entrada de datos, procesamiento, condicionales, etc.) en un proceso dado. A menudo se usan para describir algoritmos, pero se pueden usar en cualquier situación que involucre un flujo finito de acciones, con entrada y salida.

Hay otras formas de representar un algoritmo además del diagrama de flujo, como la descripción textual/verbal de los pasos (que estamos usando en este curso), el pseudocódigo o incluso la lectura del código en JavaScript u otro lenguaje de programación. Puede ver ejemplos de estas tres formas en este artículo de Alura.

En el caso del diagrama de flujo, los diferentes formatos de los cuadros de texto ayudan a definir los pasos lógicos de un proceso. Vea algunos ejemplos:

Diagrama

Descripción generada automáticamente

Las figuras predeterminadas para los tipos de operación en un diagrama de flujo son:

Inicio/Fin: Marcan el inicio o fin del procesamiento que se describe, normalmente “inicio” o “fin”, pero también se pueden utilizar en salidas de procesamiento anterior o las entradas de procesamiento posterior, como “envío de formulario”. Representado por un óvalo o rectángulo con bordes redondeados;

Entrada manual: entrada (o ingreso) manual de datos, por ejemplo, datos ingresados ​​por el usuario del sistema. Representado por un rectángulo con el lado superior derecho;

Procesamiento: bloque o conjunto de operaciones que se realizarán con los datos recibidos. Representado por un rectángulo;

Decisión: cuando el programa puede seguir más de un camino, lo que implica una decisión cuyo resultado es “sí” o “no” (los equivalentes de verdadero y falso). Representado por un diamante;

Entrada/Salida: Proceso de recepción de datos por parte del programa (entrada) o devolución del resultado de un procesamiento (salida). Representado por un paralelogramo.

Ilustremos mejor con un diagrama de flujo de ejemplo para una suma entre dos números:

Diagrama

Descripción generada automáticamente

El diagrama de flujo anterior describe:

1. El punto de partida del procesamiento;
2. El ingreso de dos datos (variables num1 y num2), cada uno a la vez;
3. Procesamiento (operación matemática entre los dos números);
4. La salida del resultado a través de la variable suma;
5. El final del procesamiento.

Podemos representar el algoritmo anterior con código JavaScript:

const num1 = 2;

const num2 = 2;

const suma = num1 + num2;

console.log(suma) //4

Veamos un ejemplo más, para un bucle que muestra números del 0 al 10 en la consola.

Diagrama

Descripción generada automáticamente

El diagrama de flujo anterior se puede traducir a código JavaScript en un for:

for (let i = 0; i <= 10; i++) {

console.log(i);

}

Hay otros símbolos que usamos en los diagramas de flujo para ilustrar otro tipo de operaciones posibles, como imprimir en papel o enviar datos a un banco.

Puede practicar con diagramas de flujo para crear algoritmos tanto para el código que ya ha escrito como para "ponerlo en papel" para ayudar a organizar la estructura de un algoritmo que está desarrollando; o incluso entrenar con otras situaciones, como hacer un pastel o pagar la cuenta de la luz.

Durante este curso estamos creando nuestros propios algoritmos de búsqueda y clasificación. Sin embargo, si investigamos la documentación de JavaScript, descubrimos que el lenguaje ya tiene disponible, listos para usar, métodos/funciones que buscan ocurrencias y ordenan arreglos, de la misma manera que lo estamos haciendo nosotros. Por ejemplo, para revisar arreglos en busca de datos específicos, podemos usar include(), find() o filter(), entre otros, y para ordenar arreglos, podemos usar el método sort().

Y entonces "¿por qué no usamos el método de JavaScript en lugar de escribir otro desde cero?" Mirémoslo desde otro ángulo: si JavaScript tiene un método/función de orden “listo para usar”, ese método seguramente usa algún algoritmo de clasificación para hacer ese trabajo. ¿Y cuál sería? ¿Es el mismo que usamos?

La verdad es que sort() usa más de un algoritmo de clasificación, dependiendo principalmente de dos cosas: 1) las características del arreglo a clasificar y 2) la implementación de JavaScript que se utiliza.

¿Cómo es eso, implementación?

JavaScript fue desarrollado como un lenguaje interpretado por los navegadores, y en base a la especificación de lo que debe seguir cada versión del lenguaje (definido por la ECMA), cada navegador implementa métodos y funciones de acuerdo a sus propios motores, es decir, cada navegador tiene su propio "motor" de interpretación de JavaScript, y la forma en que se interpreta el código "detrás de escena" en cada navegador puede ser diferente.

En el caso del método sort(), el motor de interpretación de JavaScript implementado por Mozilla en el navegador Firefox utiliza un algoritmo denominado Merge Sort, mientras que Google utiliza, en el motor de Chrome (denominado V8) alterna entre otros dos algoritmos de ordenación, el Quick Sort e Insertion Sort según el caso.

En el caso de NodeJS, dado que utiliza como base el motor V8 de Google, los algoritmos utilizados en sort() también serán los mismos.

¿Y qué casos son estos?

El método sort(), ejecutado sin ningún parámetro, interpreta todos los elementos del arreglo como cadenas y los ordena en orden alfabético ascendente a partir de la tabla Unicode. Para otros tipos de clasificación, por ejemplo numérica o descendente, es necesario pasar parámetros de comparación a sort().

Comentamos durante la clase sobre la corrida en frío o desk checking, en inglés.

Una corrida en frío es precisamente el proceso manual de comprobar todas las líneas de código y ejecutarlo paso a paso con la ayuda de, por ejemplo, papel y lápiz para anotar los valores de las variables en cada paso de cada línea. Es como compilar/interpretar un bloque de código usando el cerebro como compilador/intérprete.

Si bien los IDE y los lenguajes hoy en día cuentan con herramientas prácticas, modernas y eficientes para advertir, “depurar” y encontrar errores de código, las pruebas de tablas son una muy buena manera de estudiar la lógica de programación y comprender qué sucede en cada paso de ejecución del código, especialmente en el caso de los algoritmos que involucran bucles de repetición, sustitución de valores y variables temporales, como vimos durante la clase.

Una forma muy común de registrar valores de variables durante una corrida en frío es usando tablas de seguimiento. Por ejemplo, una tabla de prueba para un for que imprime números del 0 al 5 en la consola podría representarse mediante la siguiente tabla de prueba/tabla de seguimiento:

Texto

Descripción generada automáticamente

| **línea** | **num** | **i** | **salida** |
| --- | --- | --- | --- |
| 1 | 0 |  |  |
| 2 |  |  | 0 |
| 4 |  | 1 |  |
| 5 |  |  | 1 |
| 4 |  | 2 |  |
| 5 |  |  | 2 |
| 4 |  | 3 |  |
| 5 |  |  | 3 |
| 4 |  | 4 |  |
| 5 |  |  | 4 |
| 4 |  | 5 |  |
| 5 |  |  | 5 |
| 4 |  | 6 |  |

En tus ejercicios y pruebas, la tabla se puede hacer fácilmente con papel y lápiz, si lo prefieres.

Lo importante en las corridas en frío (con o sin usar la tabla) es revisar cada línea y ejecutar el código, anotando creaciones/reasignaciones de variables, resultados de operaciones matemáticas (si las hay), cambios en matrices y objetos, etc. De esta forma, aparecerán inmediatamente resultados inesperados en la ejecución del código.