Investigación garbaje collector

Un **recolector de basura** (del inglés *garbage collector*) es un mecanismo implícito de [gestión de memoria](https://es.wikipedia.org/wiki/Asignaci%C3%B3n_de_memoria) implementado en algunos [lenguajes de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) de tipo [interpretado](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_interpretado) o semiinterpretado.

Contexto

Cualquier [programa informático](https://es.wikipedia.org/wiki/Programa_inform%C3%A1tico) hace uso de una cierta cantidad de [memoria](https://es.wikipedia.org/wiki/Memoria_(inform%C3%A1tica)) de trabajo puesta a su disposición por el [sistema operativo](https://es.wikipedia.org/wiki/Sistema_operativo). Esta memoria tiene que ser gestionada por el propio programa para:

* Reservar espacios de memoria para su uso.
* Liberar espacios de memoria previamente reservados.
* Compactar espacios de memoria libres y consecutivos entre sí.
* Llevar cuenta de qué espacios están libres y cuáles no.

Generalmente, el programador dispone de una [biblioteca de código](https://es.wikipedia.org/wiki/Biblioteca_(inform%C3%A1tica)) que se encarga de estas tareas. No obstante, el propio programador *es responsable* de utilizar adecuadamente esta biblioteca.

Esto tiene la ventaja de que se hace un uso eficiente de la memoria, es decir, los espacios de memoria quedan libres cuando ya no son necesarios. No obstante, este mecanismo explícito de gestión de memoria es propenso a errores. Por ejemplo, un programador puede olvidar liberar la memoria de manera que, tarde o temprano, no quede memoria disponible, abortando la ejecución del programa.

Como alternativa es necesaria una gestión *implícita* de memoria, con lo que el programador no es consciente de la reserva y liberación de memoria. Esto es obligado en algunos [lenguajes de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) en los que no se maneja el concepto de memoria. Por ejemplo, en lenguajes declarativos como [Lisp](https://es.wikipedia.org/wiki/Lisp" \o "Lisp) o [Prolog](https://es.wikipedia.org/wiki/Prolog" \o "Prolog).

Funcionamiento

Cuando un lenguaje dispone de recolección de basura, el programador no tiene que invocar a una subrutina para liberar memoria. La reserva de memoria también es más o menos automática sin la intervención del programador. Por ejemplo:

* En los lenguajes orientados a objetos: se reserva memoria cada vez que el programador crea un objeto, pero éste no tiene que saber cuánta memoria se reserva ni cómo se hace esto.
* En los lenguajes declarativos: cada vez que se construye una expresión se reserva memoria (de una manera inteligente), pero el programador no es consciente de ello.

Cuando se compila el programa, automáticamente se incluye en éste una [subrutina](https://es.wikipedia.org/wiki/Subrutina) correspondiente al recolector de basura. Esta subrutina también es invocada periódicamente sin la intervención del programador.

El recolector de basura es informado de todas las reservas de memoria que se producen en el programa. Además, el compilador colabora para que sea posible llevar una cuenta de todas las referencias que existen a un determinado espacio de memoria reservado.

Cuando se invoca el recolector de basura, recorre la lista de espacios reservados observando el contador de referencias de cada espacio. Si un contador ha llegado a cero significa que ese espacio de memoria ya no se usa y, por tanto, puede ser liberado.

Naturalmente, este proceso consume un cierto tiempo en el que no se hace nada verdaderamente útil para el propósito del programa. Por tanto, no puede ser invocado con demasiada frecuencia.

En consecuencia, el único inconveniente a este mecanismo es determinar *cuándo* se tiene que ejecutar el recolector de basura. Existen varios algoritmos para hacerlo, pero el más eficiente es el primero de ellos:

* Esperar a que no quede memoria libre, y entonces, ejecutar el recolector de basura.
* Fijar un umbral de ocupación de la memoria libre y ejecutar el recolector de basura cuando se supere dicho umbral.
* Ejecutar el recolector de basura a intervalos regulares (no siempre es posible).
* Ejecutar el recolector de basura justo antes de cada reserva de memoria.
* Permitir al programador que invoque explícitamente al recolector de basura cuando quiera.

Ventajas y desventajas

Las ventajas y desventajas de este mecanismo de gestión de memoria son las opuestas al mecanismo explícito:

* El programador no puede cometer errores y queda liberado de la tediosa tarea de gestionar la memoria.
* La memoria permanece retenida durante más tiempo del estrictamente necesario.
* El recolector de basura tarda cierto tiempo en hacer su tarea y produce pausas que pueden hacer la técnica incompatible con [sistemas de tiempo real](https://es.wikipedia.org/w/index.php?title=Sistemas_de_tiempo_real&action=edit&redlink=1).

Implementación

Existe la posibilidad de implementar la recolección de basura como una biblioteca de código más, pero por norma general no es así. El propio diseño de ciertos lenguajes de programación hace necesaria la existencia del recolector de basura. Para poder implementar estos lenguajes se requieren dos actuaciones:

* Que el compilador proporcione la información necesaria para el recolector de basura (el contador de referencias).
* Que el [entorno de ejecución](https://es.wikipedia.org/wiki/Entorno_de_ejecuci%C3%B3n) o [máquina virtual](https://es.wikipedia.org/wiki/M%C3%A1quina_virtual) implemente la subrutina del recolector de basura.