Runtime: Manejo Automático de Memoria

Lic. Juan Pablo Consuegra Ayala

MatCom, UH

Manejo de Memoria

- Todo lenguaje de programación de bajo nivel requiere de una cantidad de memoria, cedida por el Sistema Operativo, para trabajar.

Los programas deben ser capaces de:

- Ubicar los objetos que está manejando en tiempo de ejecución.
- Monitorizar su estado.
- Reciclarlos cuando ya no son necesarios.
- Idealmente, compactar el espacio resultante para optimizar el rendimiento.

La gestión de memoria, dependiendo del lenguaje, puede ser de dos tipos: manual o automática.

Gestión manual de memoria

- Se asignan y liberan los recursos de memoria de forma explícita.
- El desarrollador quien debe administrar los registros asignando memoria a sus objetos mediante punteros y liberarlos cuando ya no sean necesarios.

Errores comunes

Gestión manual de memoria

- Se asignan y liberan los recursos de memoria de forma explícita.
- El desarrollador quien debe administrar los registros asignando memoria a sus objetos mediante punteros y liberarlos cuando ya no sean necesarios.

Errores comunes

- **Dangling pointers:** la memoria se libera pero quedan punteros referenciando a esa región de memoria, la cual es accedida incorrectamente.
- **Double free bugs:** el programa trata de liberar una región de memoria que ya fue liberada, y quizás incluso reservada nuevamente.
- **Memory leaks:** el programa falla en liberar memoria ocupada por objetos que se volvieron inaccesibles.

Gestión automática de memoria

Llamada también gestión implícita de memoria.

- Las tareas de reserva, monitorización y reciclaje son llevadas a cabo por el propio sistema sin la intervención del programador.
- Una subrutina se encarga de identificar y eliminar la memoria ocupada por objetos que ya no están en uso por el programa: el **Recolector de Basura**.

  En los lenguajes de alto nivel, cuando un programa es compilado, se incluyen de forma predeterminada las subrutinas que constituyen el propio recolector.

Recolección de basura por conteo de referencias

Técnica más sencilla para identificar direcciones de memoria que no están actualmente en uso.

> *Si un objeto, o dirección de memoria, no está siendo referenciado por ningún otro, se considera que no está en uso y, por tanto, puede ser liberado.*

Se mantiene una contabilidad en cada objeto de cuántas referencias apuntan al mismo.

- Normalmente se almacena en su cabecera.
- Siempre que una referencia es creada o copiada, el contador de referencias del objeto que referencia se incrementa.
- Siempre que una referencia se elimina o es sobreescrita, el contador de referencias del objeto que referencia se reduce.

Conteo de referencias

La principal ventaja de esta estrategia:

- Los objetos se liberan tan pronto como no sean referenciados,
- y lo hacen de una forma incremental, sin largas pausas de recolección y con ciclos de vida bien definidos para cada objeto.

Usado para otras optimizaciones

Si el compilador (o runtime system) sabe que un objeto particular solo tiene una referencia, y esa referencia se pierde a la misma vez que un objeto similar es creado, entonces puede reemplazar la operación con una mutación del objeto original.

Ejemplo: `str ← str + "a"`

Desventajas de conteo de referencias

Cada objeto necesita un tamaño ligeramente mayor para almacenar el recuento de referencia.

El campo del objeto que se utiliza para llevar el conteo posee un tamaño limitado . . .

- El sistema puede colapsar si el número de referencias posibles a dicho objeto es ilimitado.

Cada modificación de un puntero implica una operación . . .

- Aumenta el tamaño del código.
- Aumenta la demanda de ancho de banda de memoria.
- Puede derivar en una grave penalización de rendimiento (especialmente en entornos multihilo donde las actualizaciones de conteo de referencia requieren sincronización).

Pero hay problema mayor . . .

Dos objetos X y Y forman un **conjunto mutuamente dependiente** si:

- X contiene una referencia al objeto Y.
- Y contiene una referencia al objeto X.

Qué ocurre si alguno de los objetos forma parte de una estructura de datos cíclica?

Pero hay problema mayor . . .

Dos objetos X y Y forman un **conjunto mutuamente dependiente** si:

- X contiene una referencia al objeto Y.
- Y contiene una referencia al objeto X.

Qué ocurre si alguno de los objetos forma parte de una estructura de datos cíclica?

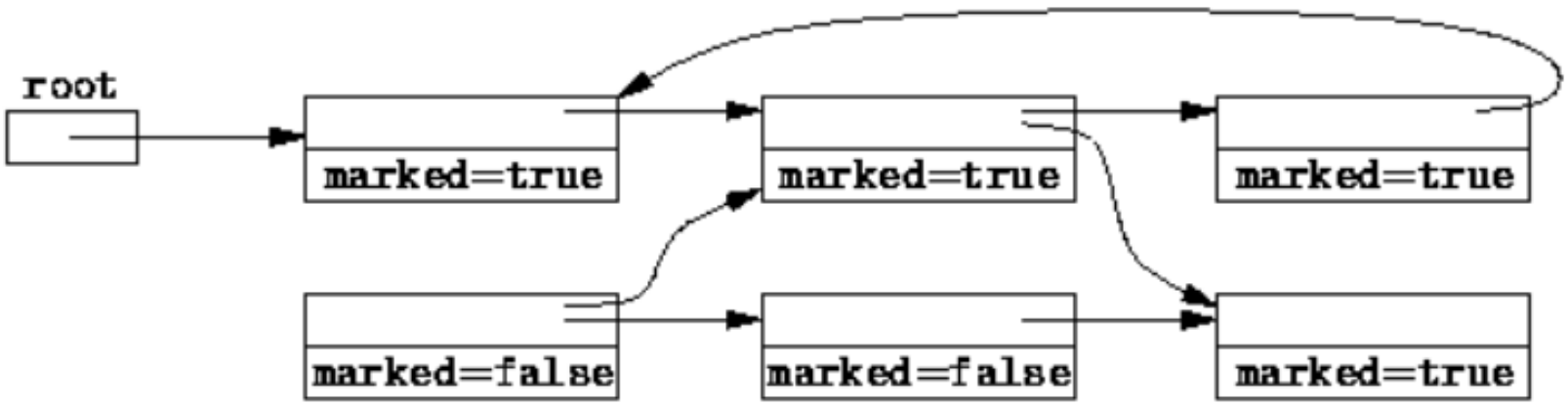
- Aunque todo el conjunto quede en desuso y no sea necesario en el programa, su conteo de referencias siempre mostrará al menos una dependencia, por lo que el recolector no puede actuar sobre él!!!

Recolección de basura por marcado y barrido (Mark-Sweep)

En esencia, se realizan dos operaciones básicas:

- **"marcar"** qué objetos son inalcanzables.
- **"barrer"** la memoria que ya no es necesaria para devolverla nuevamente al programa.

Fase de marcado



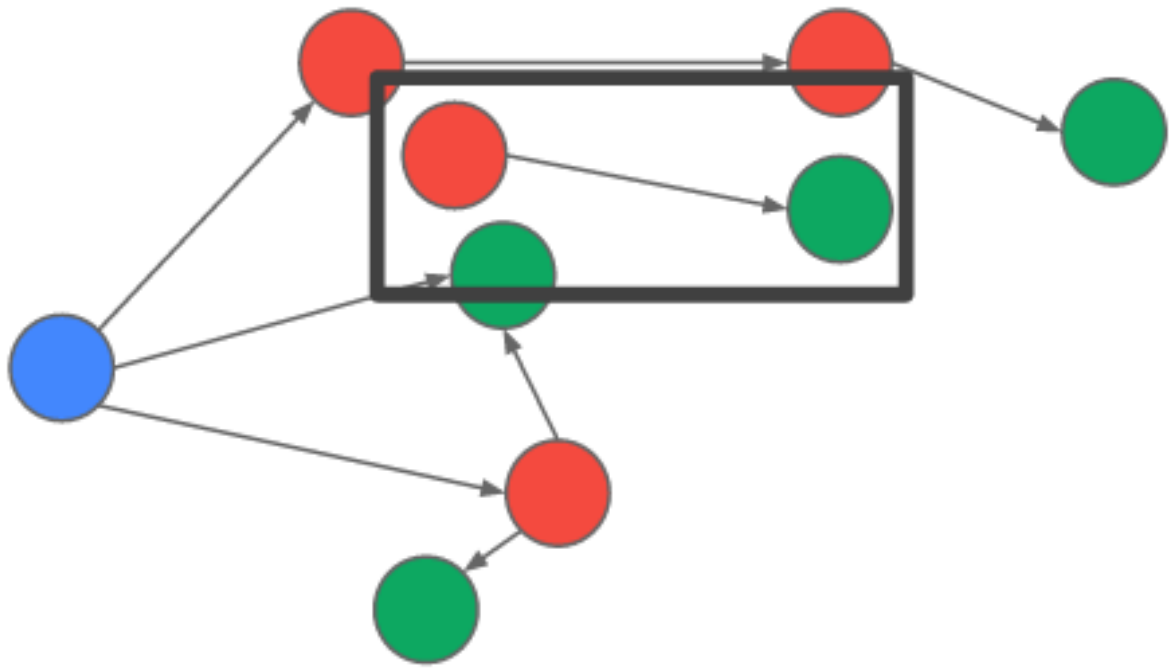
- Aquellos objetos referenciados por las variables globales (*raíces*), se consideran **objetos directamente accesibles**.
- Aquellos objetos referenciados por algún otro objeto accesible (*directa o indirectamente*), se consideran **objetos indirectamente accesibles**.

Representación

- **Bit de Marca (Bit Mark):** Se asocia al puntero del objeto un indicador de un bit para indicar si tiene al menos una referencia apuntándole.
- **Mapeado de bits (Bitmap Marking):** Se almacena la correspondiente marca de bit de un objeto en un rango de memoria separado que constituye un diccionario

Fase de barrido

Eliminar aquellos objetos que han quedado aislados y que, por tanto, son considerados como desechables.

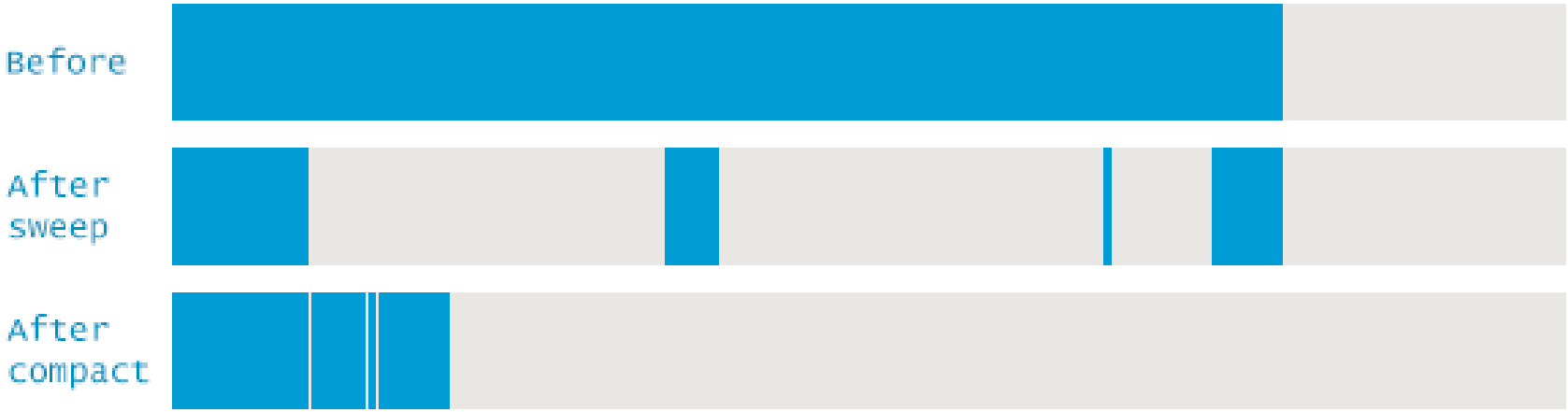


Compactación

Tras el reciclaje de objetos la memoria del sistema puede terminar muy fragmentanda.

- Ralentiza futuras asignaciones.

Mover y reubicar los objetos para eliminar el espacio muerto que ha podido quedar entre ellos.



Notas finales

Existen otras estrategias y optimizaciones a aplicar . . .

- **Escape analysis:** Convertir heap allocations a stack allocations (=> determinar si un objeto creado dentro de una función es accesible o no fuera de ella).
- . . .

Cómo añadir recolector de basura al lenguaje?

Bootstrapping.