Investigación Multidisciplinar II

Proyecto: Algoritmo de reconstrucción TwIST en C++

Lenguaje: C++

Kareth Marcela León López **Código**: 2158724

Óscar Enrique Hurtado Camacho **Código**: 2158725

Informe: Análisis de memoria

Análisis con Valgrind

Con el objetivo de hacer una revisión del uso de memoria en el proyecto de clase titulado: "***Algoritmo de reconstrucción TwIST en C++"*,** se hace uso del comando *memcheck* de valgrind para detectar:

- El uso de la memoria sin inicializar
- Lectura / escritura de la memoria después de haber sido liberada
- Lectura / escritura al final de la asignación con malloc
- Lectura / escritura de áreas inapropiadas en la pila
- Las pérdidas de memoria – *memory leaks* – donde los punteros a memoria asignada a bloques se pierden para siempre

Comando: *valgrind - -tool = memcheck ./main.o*

Captura de pantalla:

```
==22578==
==22578==
==22578== HEAP SUMMARY:
==22578==     in use at exit: 417,816 bytes in 16 blocks
==22578==   total heap usage: 2,071 allocs, 2,055 frees, 34,097,288 bytes allo
cated
==22578==
==22578== LEAK SUMMARY:
==22578==    definitely lost: 98,304 bytes in 3 blocks
==22578==    indirectly lost: 0 bytes in 0 blocks
==22578==      possibly lost: 0 bytes in 0 blocks
==22578==    still reachable: 319,512 bytes in 13 blocks
==22578==         suppressed: 0 bytes in 0 blocks
==22578== Rerun with --leak-check=full to see details of leaked memory
==22578==
==22578== For counts of detected and suppressed errors, rerun with: -v
==22578== Use --track-origins=yes to see where uninitialised values come from
==22578== ERROR SUMMARY: 90112 errors from 14 contexts (suppressed: 0 from 0)
kareth@Kamaleon:~/Dropbox/GIT HUB/TwIST$ git status
```

Fig 1. Captura de pantalla de los resultados a la llamada a *valgrind*.

En texto:

```
==22578== HEAP SUMMARY:
==22578==     in use at exit: 417,816 bytes in 16 blocks
==22578==   total heap usage: 2,071 allocs, 2,055 frees, 34,097,288 bytes allocated
==22578==
==22578== LEAK SUMMARY:
==22578==    definitely lost: 98,304 bytes in 3 blocks
==22578==    indirectly lost: 0 bytes in 0 blocks
==22578==      possibly lost: 0 bytes in 0 blocks
==22578==    still reachable: 319,512 bytes in 13 blocks
==22578==         suppressed: 0 bytes in 0 blocks
==22578== Rerun with --leak-check=full to see details of leaked memory
==22578==
==22578== For counts of detected and suppressed errors, rerun with: -v
==22578== Use --track-origins=yes to see where uninitialised values come from
==22578== ERROR SUMMARY: 90112 errors from 14 contexts (suppressed: 0 from 0)
```

Resultados

1. **HEAP SUMMARY**
   a. Total heap usage: 2,071 allocs, 2,055 frees, 34,097,288 bytes allocated. En el código se liberan 2055 bloques de 2071 usados, por lo cual en el momento de finalizar la aplicación, se encuentra que aún existen 16 bloques de 417,816 bytes sin liberar.

2. **LEAK SUMMARY (Fuga de memoria)**
   a. Pérdida definitiva: 98, 304 bytes en 3 bloques.
   En el proyecto se presentan fugas de memoria indicando que no existen apuntadores a espacios de memoria de variables creadas.

Análisis con Callgrind

A partir del uso de la herramienta Callgrind, se observa que las llamadas del archivo ejecutable del proyecto de concentran en la función *twist* con el **94.51%**.

Luego, dentro de esta función (*twist*), el procesamiento se concentra en las funciones de multiplicación de matrices, Af y AT, que significan la mayor parte de la complejidad computacional debido a las dimensiones de las matrices, las cuales representan el **80.9%** y **13.21%** de las llamadas, respectivamente. Particularmente, para la función Af, se presenta el mayor porcentaje de llamadas dado a el número de filas de la matriz que recibe la función es mayor que el número de columnas, esto implica que se harán mayor número de multiplicaciones con respecto a su transpuesta en la función AT.

El resto de las llamadas a diferentes funciones representan menos del **2%.** En la

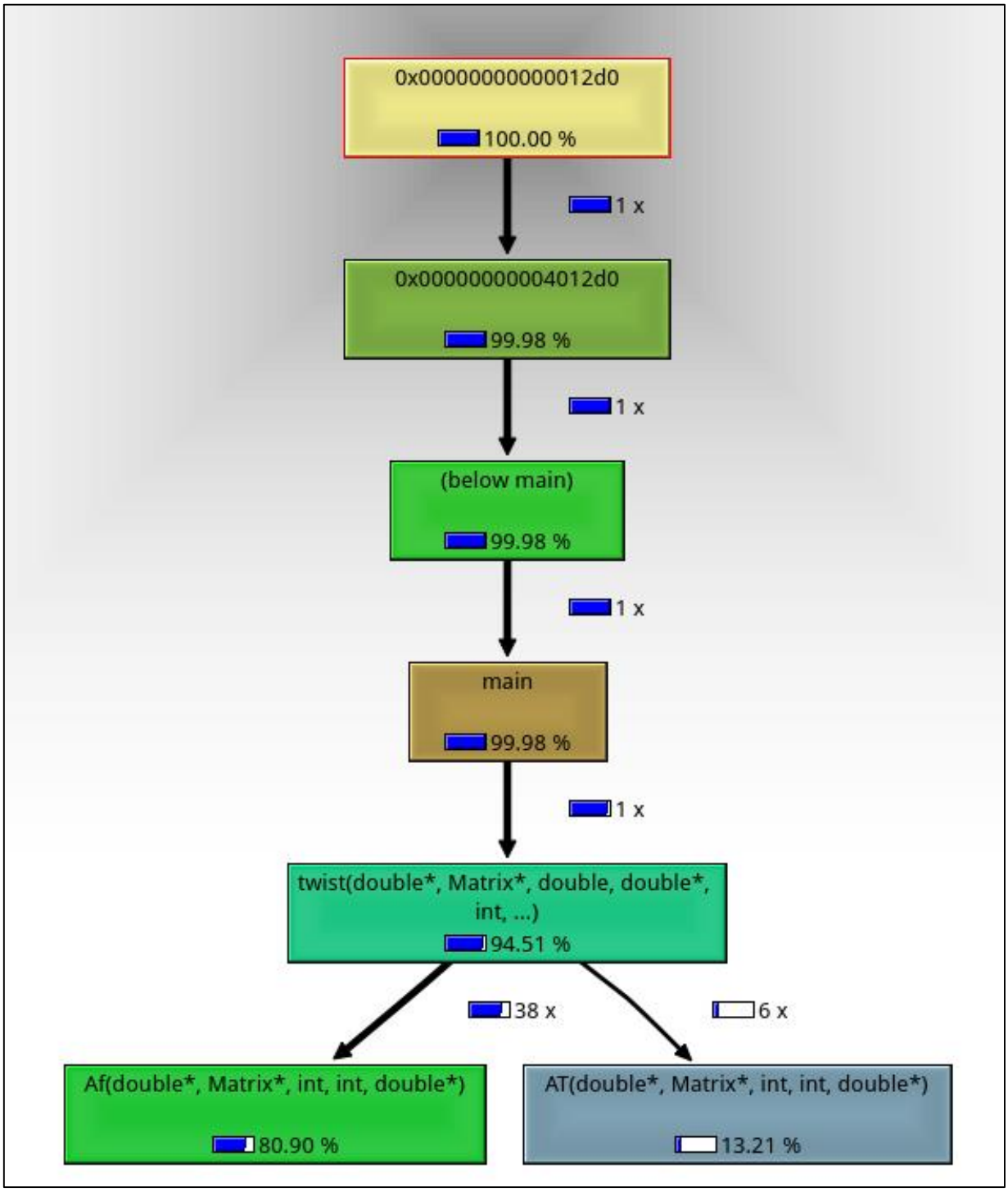

Fig 2. Gráfico de llamadas (Call graph). La función ***twist*** presenta el mayor porcentaje de llamadas del proyecto.

Conclusiones

- En el proyecto implementado, se hizo uso de las herramientas Git, Valgrind y Make para el desarrollo del mismo.
- Fue posible detectar y corregir errores a partir del uso de herramientas de depuración.

- Se detectaron fugas de memoria y liberaron bloques de memoria a partir del análisis realizado con la herramienta de Valgrind, sin embargo, no se logró corregir todos los errores. Por otro lado, se tuvieron inconvenientes con el uso de Valgrind en medio de la realización del proyecto debido a la necesidad de archivos ejecutables, es por esto, que solo se presenta solo un análisis con la herramienta Valgrind durante el desarrollo del proyecto.