Informe Caldep

Santiago Aníbal Carrillo Torres
2259465
Universidad del Valle (sede Tuluá)
Riofrio, Valle del Cauca
santiago.carrillo@correounivalle.edu.co

Edwar Yamir Forero Blanco
2259664
Universidad del Valle (sede Tuluá)
Tuluá, Valle del Cauca
edwar.forero@correounivalle.edu.co

Juan Eduardo Calderon Jaramillo
2259671
Universidad del Valle (sede Tuluá)
Tuluá, Valle del Cauca
juan.eduardo.calderon@correounivalle.edu.co

***Abstract*— Para la realización de este proyecto se realizaron 2 soluciones, una ingenua la cual tiene un rendimiento computacional bastante bajo, es decir que su crecimiento es exponencial y muy limitado al momento de encontrar las soluciones para distintos equipos; la óptima permite que las soluciones puedan ser creadas de la mejor manera, debido a que su costo computacional es mucho más bajo y por lo tanto permite mejorar el rendimiento. En ambas se encuentran las explicaciones del código, los análisis y la muestra de las pruebas con distintos números de equipos. Además, se presentan conclusiones respecto al proyecto realizado**

I. INTRODUCTION

El torneo llamado CalDep busca generar un calendario deportivo el cual tiene un formato de enfrentamiento de todos contra todos. El torneo posee ciertas restricciones para tener en cuenta, las más importantes son minimizar los recorridos para cada equipo y tener un mínimo y máximo de partidos consecutivos dependiendo si son locales o visitantes.

Para llevar a cabo la solución del proyecto se plantean dos soluciones, una ingenua y otra optimizada, ambas deben generar un calendario deportivo en el cual se describa las fechas del torneo y los respectivos enfrentamientos tanto de visitante como de local para cada equipo.

Este proyecto es realizado en el lenguaje de java, ya que por el conocimiento que posee el equipo de trabajo, es la opción que ofrece mayor rendimiento y facilidad al momento de ejecutar el desarrollo

II. SOLUCIONES

A. Solucion Ingenua

Esta solución tiene como finalidad suplir meramente la necesidad del torneo, es decir no se tienen en cuenta en gran medida el factor de rendimiento computacional.

Para realizar esta solución se tuvieron en cuenta cuatro puntos importantes: generación de partidos, posibles permutaciones de las filas, suma de los recorridos y máximos y mínimos de partidos consecutivos de localías y visitas. A continuación, se muestra una explicación y análisis de cada uno de los puntos:

Generación de partidos: Para este punto se usaron dos tipos de estructuras de datos, **List O(1)** que guardará todos los equipos inscritos con una función **"guardarEquipos"** para luego ser usados en la elección de enfrentamientos y **Array O(n^2)** de dos dimensiones que tiene como finalidad guardar cada uno de los enfrentamientos de los equipos durante todas las fechas del torneo. Este arreglo se completará en la función **"OrgFechas"** para la primera mitad de las fechas, para que luego en "Vuelta" se asigne la otra mitad de los partidos.

Permutaciones: La función recursiva **"permutaciones" (O (2*(n-1))!)** que a partir del número de fechas y el arreglo de partidos del torneo, intercambiará cada una de las filas para esa misma fecha a partir de la recursividad y luego de ella, se realiza el mismo cambio, para que el arreglo no pierda su originalidad y sea posible realizar todas las permutaciones.

Sumas Recorridos, Mínimo y Máximo: Los métodos "**SumaRecorridos" O(n^2)** y "**MinYMax**" **O(n^2)** serán llamados para cada arreglo permutado y hallar la mejor opción posible.

Análisis:

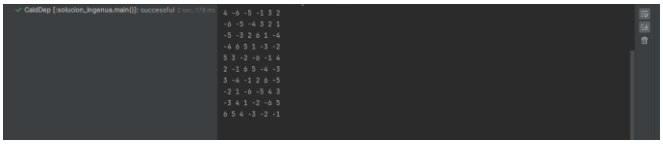
La listas enlazadas (O(n)) se eligen, para que lleve un control hacía que equipos se le están asignando los partidos, ya que luego de realizar la asignación a cierto número de fechas, este dato se remueve y se vuelve a agregar, para que de esta forma se ubique de último, y así no tener problemas en cuanto a fechas repetidas durante la primer mitad.

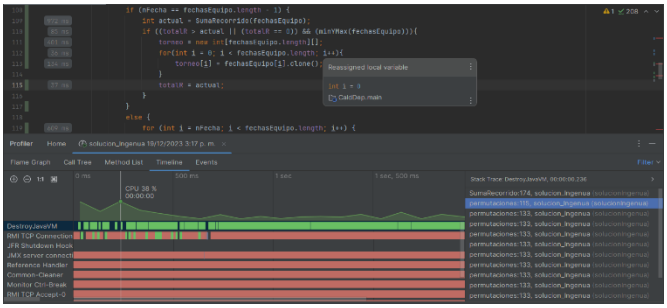
En cuánto a los arreglos (O(n^2)), se utilizaron, debido a que es una estructura bastante fácil de comprender y de modificar para este caso, como lo puede ser para la permutación de filas. Además, que en cuánto a los costos en espacio al usar enteros, se tiene que su espacio es de 32 bytes por dato en el arreglo, comparado a un List que posee hasta 32*3 bytes inicialmente.

En cuanto a la complejidad del programa, tendría un total de **O(n!)**. Indica una complejidad extremadamente alta, lo que significa que significa un valor exponencial al momento de

aumentar el número de equipos. Para este caso la eficiencia en tiempo para equipos de tamaño mayor o igual a 8 no es posible realizar el cálculo en un tiempo prudente.

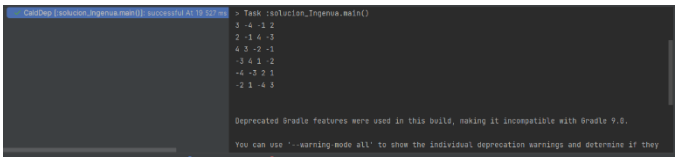
En cuanto a los tiempos de ejecución a medida que se aumenta el número de equipos va aumentando considerablemente el tiempo, aquí se muestran algunos ejemplos: Para seis equipos





Para este caso se realizó una prueba para 6 equipos con un máximo de 5 partidos consecutivos y 1 partido mínimo donde se tuvo un pico de uso CPU del 38% antes de los 500 ms y un tiempo de ejecución total de 2 segundos. El lugar donde se nota el pico de ejecución es donde más se reasignan variables, mientras se realizan las permutaciones. Esto puede ser causa de que es el inicio de la ejecución y, por lo tanto, la reasignación de variables es mayor debido a que el arreglo está ordenado de una forma poco óptima en cuento a sus recorridos tomando, esto ocasiona que la computadora necesite más uso de recursos. El número de permutaciones se tienen un aproximado de 10!.

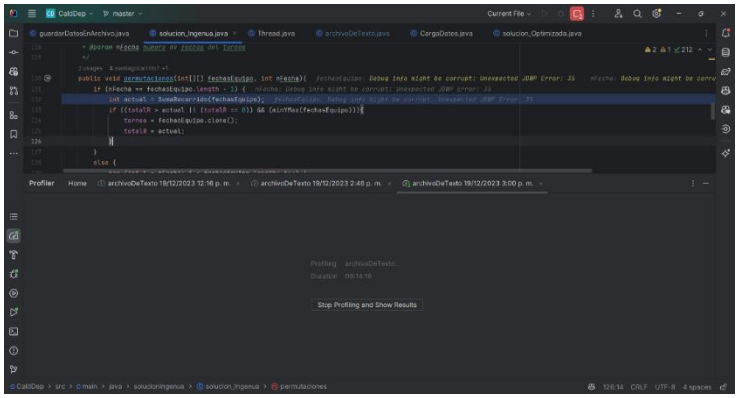
Pruebas de 4 equipos:



La prueba con 4 equipos, el tiempo de ejecución es demasiado bajo, ya que, el número de combinaciones es bajo comparado a un torneo con más equipos y cercano a (6!) permutaciones.

La prueba para 8 equipos no fue posible realizarla, debido a que su número de permutaciones es de (14!). Es decir que es extremadamente alto y con un crecimiento exponencial, lo que significa que la solución de este problema en un tiempo prudente es casi imposible de realizarse.

En cuanto el costo de memoria de acuerdo con las estructuras de datos, obtenemos que:



- Variables Primitivas:
  - private int[][] torneo: Esta variable almacena una matriz de enteros. El tamaño de la matriz es fila x columna. El costo en memoria depende de la cantidad de elementos en la matriz.

    Equipo de (4): 4*4*4 = 64 bytes

    Equipo de (6): 6*6*4 = 96 bytes

    Equipo de (8): 8*8*4 = 256 bytes
  - private int totalR=0: Esta variable es un entero y ocupa 4 bytes en memoria.
- Estructuras de Datos:
  - private final int[][] enfrent;: Es una matriz que almacena enteros y tiene el mismo tamaño que torneo.

    Equipo de (4): 4*4*4 = 64 bytes

    Equipo de (6): 6*6*4 = 96 bytes

    Equipo de (8): 8*8*4 = 256 bytes
  - private final int columna;: Almacena el número de columnas en la matriz, es un entero y ocupa 4 bytes.
  - private final int min, max;: Son enteros que representan los valores mínimo y máximo, cada uno ocupa 4 bytes.
  - List<Integer> teams: Una lista que almacena enteros, su tamaño depende del número de equipos.

    Equipo de (4): 4*4 = 16 bytes

    Equipo de (6): 6*4 = 24 bytes

    Equipo de (8): 8*4 = 32 bytes
- Métodos y Operaciones:
  - List<Integer> teams = new ArrayList<>();: Se crea una lista de enteros, ocupará memoria proporcional al número de equipos.

Equipo de (4): 4*4*4 = 64 bytes

Equipo de (6): 6*6*4 = 96 bytes

Equipo de (8): 8*8*4 = 256 bytes

- int[][] recorridos;: Es una matriz de enteros que se inicializa con
- matrizDistancias, su tamaño es columna x columna. Las variables locales utilizadas dentro de los métodos también ocupan memoria.

Conclusión de la solución Ingenua:

Esta solución demuestra su eficacia en la generación de calendarios al cumplir con los requisitos establecidos para el número máximo y mínimo de partidos consecutivos. Aunque su enfoque busca un desplazamiento que, si bien no es óptimo en su totalidad, se acerca considerablemente a la optimización deseada. Es evidente que el algoritmo presenta diversas limitaciones debido a su complejidad computacional, lo cual restringe su aplicabilidad únicamente a equipos de tamaño 6 o menores. A pesar de estas limitaciones, la solución logra cumplir con los criterios esenciales, destacando la importancia de considerar alternativas más eficientes para adaptarse a equipos de mayor tamaño sin comprometer la calidad del resultado.

B. *Solucion Optimizada*

En esta solución se busca realizar lo mismo que en la solución ingenua, pero ese necesario mejorar la complejidad computacional, pues se desea lograr una ejecucion mas eficiente en cuanto a tiempo y costo de espacio, para llevarlo a cabo se tuvieron en cuenta varios factores a tener en cuenta:

- Primero: Se tenía una matriz 2D para almacenar las fechas del torneo. Esto es eficiente para acceder a los elementos, pero si necesita realizar operaciones como insertar o eliminar elementos, una Lista de Listas podría ser más eficiente.
- La segunda opción era implementar un algoritmo genético el cual se usa para resolver este tipo de problemas del viajante. Esta sería una buena opción la cual nos permitiría mejorar el rendimiento en cuánto a costos computacionales más bajos, pero su debilidad se podría encontrar al momento de realizar las operaciones dadas, por lo que la realización de este torneo sería
- La tercera es aplicar programación lineal, la cual ayude a reducir esta complejidad
- La cuarta seria utilizar librerías las cuales ya está optimizadas y cuentan con estructuras que puede ayudar solucionar este tipo de problemas, unas de ellas pueden ser: IBM CPLEX Optimizer o Apache Commons Math.
- La última opción y la elegida fue limitar el número de permutaciones que se realizan por cada n equipos. En este caso, se realizan permutación de 4^n donde n es el número de equipos que participaron del torneo. Para este caso las permutaciones se realizan con un número random entre 0 y el 2*(n-1) veces, que se encargará de permutar las filas.

  Este programa también tendría una generación de partidos aleatoriamente, que luego será usada para realizar cierto tipo de modificaciones.

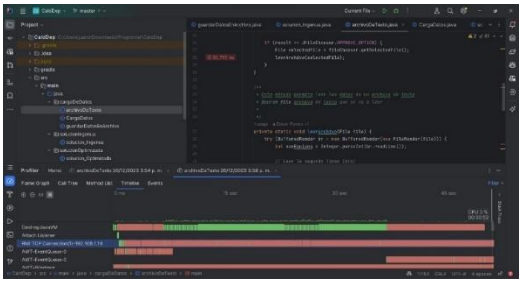
  La estructura de datos elegida sería la misma que para la ingenua, pero se busca un cambio en cuánto a como será implementada, ya que el cambio que se realiza es en eliminar el método recursivo en **“permutaciones”** y diseñar un Random entre 0 y 2*(n-1) que dos variables tomarán y se encargarán de permutarse entre si.

Inicialmente tendría el costo computacional de O(n^n) donde n es el número de equipos, debido a que se realizan 4^n veces, lo que significa que, al momento de realizar un torneo con 4 equipos sería 4^4 la mayor complejidad.

En cuanto al espacio en memoria este es mucho más bajo, debido a que no se usa la recursividad y por lo tanto los llamados hacia un mismo método en memoria serán mucho menores.

Aquí se muestra un costo de memoria de acuerdo con las estructuras de datos, obtenemos que:

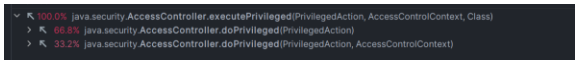
- Matrices (torneo y enfrent): Cada elemento de estas matrices es un entero, que ocupa 4 bytes. Por ejemplo, para una matriz de tamaño fila x columna, el costo de memoria sería aproximadamente fila x columna x 4 bytes.
- List (teams): Cada elemento en la lista ocupa una referencia (generalmente 4 bytes) más el tamaño del entero (4 bytes). Si tienes columna elementos en la lista, el costo sería aproximadamente columna x 8 bytes.
- Matriz recorridos: Similar a las matrices anteriores, el costo sería aproximadamente columna x columna x 4 bytes.



- Variables Primitivas (como min, max, limitePermutaciones, totalR, etc.): El costo total dependerá del tamaño de estas variables, pero en general, su impacto en el consumo de memoria es bajo.
- Variables Locales (como fila1, fila2, permutacion, actual, posicionEqui, posicionEqui2, suma, contPos,

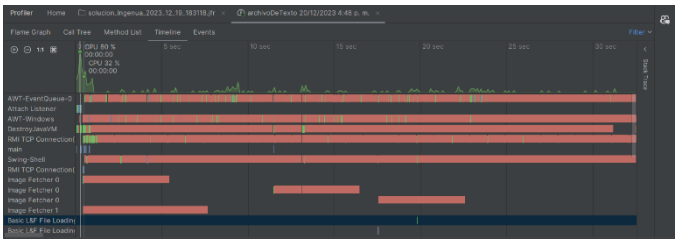
contNeg, etc.): Estas variables locales ocupan espacio en la pila y su impacto en la memoria es efímero, ya que se liberan después de que el método termina de ejecutarse.

- Esto es un ejemplo de lo consumido al ejecutar un problema con 8 equipos



En este caso se mostrarán algunos ejemplos con la ejecución de esta solución:

```
8
1
7
-8 -7 -6 -5 4 3 2 1
-2 1 -8 -7 -6 5 4 3
5 -8 -7 -6 -1 4 3 2
3 -4 -1 2 8 7 -6 -5
4 -6 -5 -1 3 2 8 -7
2 -1 8 7 6 -5 -4 -3
-5 8 7 6 1 -4 -3 -2
-6 -3 2 8 7 1 -5 -4
7 5 4 -3 -2 -8 -1 6
-4 6 5 1 -3 -2 -8 7
8 7 6 5 -4 -3 -2 -1
-3 4 1 -2 -8 -7 6 5
-7 -5 -4 3 2 8 1 -6
6 3 -2 -8 -7 -1 5 4
```



Matriz 8x8:

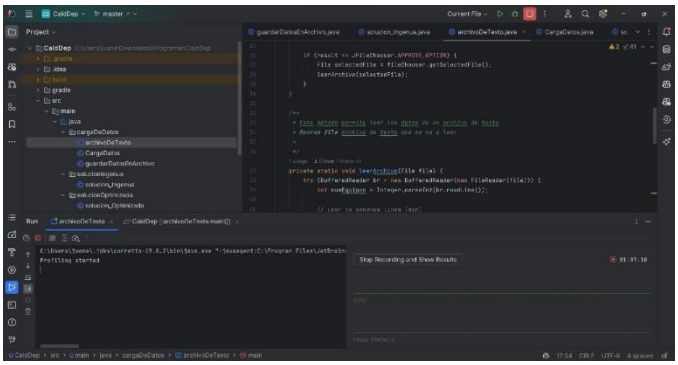
En cuánto, la solución óptima permite se tiene que el pico más alto del cpu con un 70% de su utilidad, luego se realizó al inicio de programa, pero luego se presenta una estabilidad y los picos que se muestran dentro del programa, son las representaciones de las asignaciones hacia las mejores soluciones, ya que se están cambiando variables tanto de recorridos, como el arreglo que será mostrado por el programa.

Matriz 12x12:

Para la matriz 12x12 también se es posible encontrar las soluciones, para este caso se realizan cerca de 4^12 posibles permutaciones y tuvo un tiempo de 45 segundos, donde se obtienen la mejor y esta es la que se muestra en pantalla. Para este caso, los picos de ejecución demuestran cambios o reasignaciones de variables, lo que significa que gran parte del tiempo de ejecución este algoritmo, estuvo cambiando valores en sus variables de total y torneo.

Matriz 20x20:

```
12
1
11
7 -12 -11 -10 -9 -8 -1 6 5 4 3 2
-11 -9 -8 -7 -6 5 4 3 2 12 1 -10
4 -6 -5 -1 3 2 12 11 10 -9 -8 -7
-8 -3 2 12 11 10 9 1 -7 -6 -5 -4
-10 -7 -6 -5 4 3 2 12 11 1 -9 -8
-9 -5 -4 3 2 12 11 10 1 -8 -7 -6
-6 10 9 8 7 1 -5 -4 -3 -2 -12 11
-2 1 -12 -11 -10 -9 -8 7 6 5 4 3
12 11 10 9 8 7 -6 -5 -4 -3 -2 -1
-12 -11 -10 -9 -8 -7 6 5 4 3 2 1
3 -4 -1 2 12 11 10 9 -8 -7 -6 -5
5 -8 -7 -6 -1 4 3 2 12 11 -10 -9
6 -10 -9 -8 -7 -1 5 4 3 2 12 -11
-3 4 1 -2 -12 -11 -10 -9 8 7 6 5
-7 12 11 10 9 8 1 -6 -5 -4 -3 -2
2 -1 12 11 10 9 8 -7 -6 -5 -4 -3
11 9 8 7 6 -5 -4 -3 -2 -12 -1 10
-5 8 7 6 1 -4 -3 -2 -12 -11 10 9
9 5 4 -3 -2 -12 -11 -10 -1 8 7 6
10 7 6 5 -4 -3 -2 -12 -11 -1 9 8
-4 6 5 1 -3 -2 -12 -11 -10 9 8 7
8 3 -2 -12 -11 -10 -9 -1 7 6 5 4
```



Para este caso el tiempo de respuesta y ejecución, es muy alto por lo que, es uno de los límites que posee este programa.

Conclusion Solucion Optima.

Para mejorar la solución ingenua y asi convertirla en optima se puede tomar varios caminos, pero se debe tener gran conocimiento para logra mejorar la complejidad computacional y que asi permita crear calendarios para grupos de equipo de gran tamaño manteniendo los limites especificados. La manera elegida, aunque no lo lleva a una solución perfectamente optima, si da una solución acercándose a lo óptimo. Pues se buscó

reducir el número de permutación para asi no sobrecargarlas y que pudiera dar una solución acercándose a lo óptimo y cumpliendo con los límites de máximos y mínimos partidos seguidos

Durante las investigaciones, se ha tenido en cuenta que llegar a la solución óptima es una tarea un tanto difícil en cuánto a un costo computacional bajo se refiere, por lo que se tiene en cuenta que los recorridos en esta solución serán los mejores en cuantos al número de permutaciones que se realizaron, pero estas no significan que sean las mejores dentro de la totalidad de las posibilidades que se tienen para cada equipo durante todas las fechas jugadas, ya que se es casi imposible realizar todas las permutaciones posibles.