Informe Trabajo Final CalDep

Autor(es):
George Eiver Chamorro Patiño (2259521), Jhojan Serna Henao (2259504), Faber Alexis Solis Gamboa (2259714)

Resumen- Este documento es un informe que tiene como objetivo especificar y demostrar los aspectos claves de los resultados y experiencia académica obtenidos al desarrollar el programa. El programa por desarrollar consiste en un problema de scheduling con la generación de calendarios para programar partidos.

Análisis temporal y espacial.

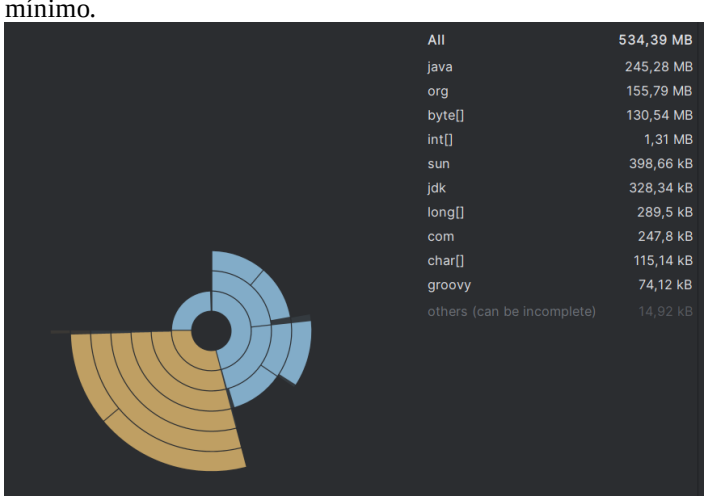
Temporal:
Este programa genera un número de permutaciones n (número de equipos) aleatoriamente, el código utiliza el método de Fisher-Yates para permutar un arreglo.
Este algoritmo tiene una complejidad O(n) dado que en el peor caso intercambia cada elemento una vez, lo cual resulta demasiado conveniente para permutar un arreglo ya que cada número tiene la misma esperanza matemática de terminar en cualquier posición.

Otras estructuras como arreglos, booleanos y enteros son utilizadas en este proyecto. Por si solas no presentaron una complejidad mucho mayor, y fueron utilizadas específicamente para no aumentar mucho más la complejidad, pero debido a que para algunas validaciones se deben recorrer las permutaciones dichas validaciones terminan con una complejidad de O(n). Además, se tienen otras validaciones para la mitad del calendario, estas validaciones utilizan bucles anidados que tiene una complejidad de O(n2).

Espacial:
El espacio en memoria está en su mayoría ocupado por arreglos y matrices, esto debido a la gran cantidad de permutaciones generadas para cumplir con las restricciones, el espacio generado es proporcional a n2 para las matrices y los arreglos al tratarse solo con datos enteros su espacio es proporcional a 32 bits por el número de equipos.

Class	Count	Shallow	Retained
byte[]	2.486.011	130,54 MB	129,58 MB
org.gradle.internal.logging.events.StyledTextOutputEvent	2.394.954	95,8 MB	440,67 MB
java.lang.String	2.483.762	59,61 MB	178,7 MB
java.lang.Object[]	2.422.850	58,91 MB	292,17 MB
java.util.ArrayList	2.404.732	57,71 MB	347,99 MB
java.util.concurrent.ConcurrentLinkedQueue$Node	2.398.555	57,57 MB	498,26 MB
org.gradle.internal.logging.events.StyledTextOutputEvent$S	2.394.956	57,48 MB	229,91 MB
java.util.concurrent.ConcurrentHashMap$Node	51.419	1,65 MB	4,09 MB
int[]	1.596	1,31 MB	1,31 MB
java.lang.Class	14.605	1,05 MB	7,03 MB
java.lang.reflect.Method	9.900	871,2 kB	1,8 MB
java.util.HashMap$Node	22.790	729,28 kB	1,69 MB
java.util.HashMap$Node[]	5.247	526,79 kB	2,16 MB
java.util.concurrent.ConcurrentHashMap$Node[]	1.216	515,21 kB	4,67 MB
java.lang.Object	31.251	500,02 kB	499,98 kB
org.codehaus.groovy.runtime.metaclass.MetaMethodIndex$E	8.572	480,03 kB	979,79 kB
java.lang.ref.WeakReference	12.017	384,54 kB	384,42 kB
java.lang.ref.SoftReference	8.391	335,64 kB	240,26 kB
java.lang.Class[]	12.286	335,57 kB	335,46 kB
long[]	3.464	289,5 kB	289,23 kB
java.lang.invoke.MemberName	6.540	261,6 kB	483,41 kB
java.lang.invoke.LambdaForm$Name	8.091	258,91 kB	502,18 kB
org.codehaus.groovy.reflection.CachedMethod	3.899	249,54 kB	1,51 MB
java.lang.invoke.MethodType	5.934	237,36 kB	677,46 kB
java.util.HashMap	4.369	209,71 kB	2,31 MB
java.lang.invoke.MethodType$ConcurrentWeakInternSet$\	5.934	189,89 kB	189,89 kB

En cuanto a las variables locales y estructuras de datos utilizadas para controlar los bucles tienen un espacio muy pequeño de memoria, lo cual genera un costo de impacto mínimo.



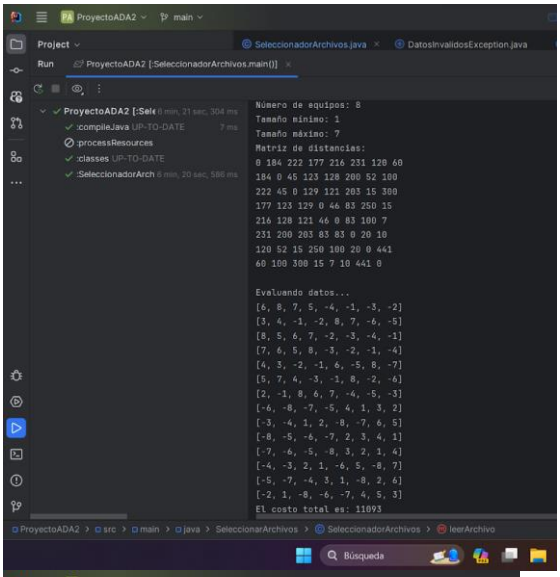
El impacto en la memoria dependerá en su gran mayoría en cómo se comporten los métodos al manejar el número de equipos, dado que para valores muy altos se tendrá un costo mucho mayor.

Análisis de cercanía al óptimo:

Este programa en conclusión resultó con una complejidad mucho mayor a la esperada.

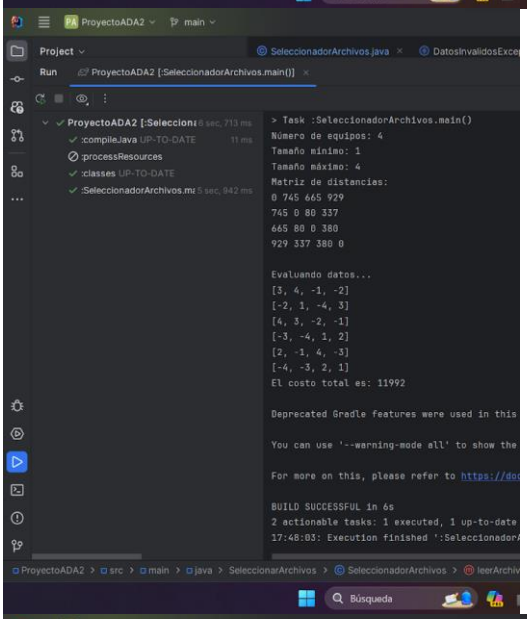
No se pudo optimizar más el código sin afectar en gran medida la lógica de su estructura, para que siguiera cumpliendo todas las restricciones.

Se esperaba que limitar las posibilidades de permutaciones sería una estrategia efectiva para lidiar con un número significativamente alto de equipos. No obstante, la implementación de esta restricción incrementó notablemente la complejidad del algoritmo, generando complicaciones que comprometen su eficiencia en términos de optimización. Este aumento en la complejidad dificulta considerablemente la propuesta inicial, resultando en un algoritmo que, aunque es óptimo para entradas no muy grandes de equipos, no genera el calendario con el menor costo.



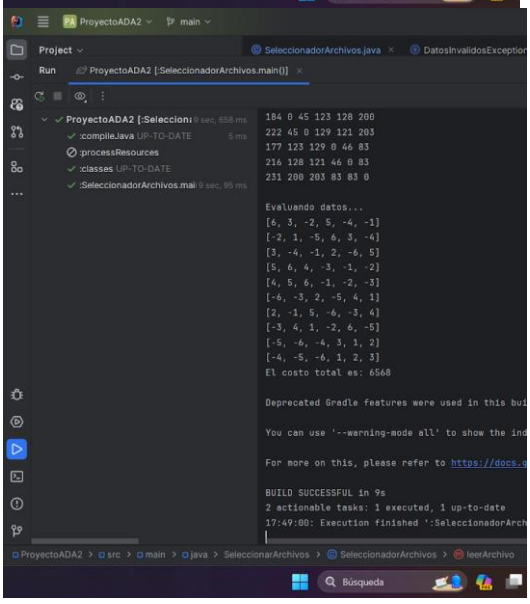
Detalles de implementación:

En la gestión integral de la programación de eventos, se eligió una estrategia que se basa en crear un conjunto grande de posibles combinaciones de horarios de manera aleatoria. La idea principal es probar diferentes opciones para encontrar la mejor programación posible con el menor costo. Este enfoque implica la utilización de la aleatoriedad para generar combinaciones de horarios, y luego se someten a un riguroso proceso de validación.



Las combinaciones de horarios se crean utilizando el algoritmo de Fisher-Yates, lo que permite una gran variabilidad en la disposición de los eventos. Sin embargo, para asegurar que los resultados sean coherentes y válidos, se realizan verificaciones exhaustivas. Estas incluyen confirmar que las combinaciones de horarios son válidas, asegurándose de que ningún evento se solape consigo mismo, y validando que la programación resultante cumple con ciertos criterios establecidos.

Este enfoque no solo busca diversificar la disposición de los eventos, sino que también incorpora medidas para garantizar que la programación generada sea coherente y válida. La combinación de aleatoriedad controlada y validaciones rigurosas crea una estrategia sólida y adaptable para enfrentar el desafío de la programación, ofreciendo flexibilidad y la capacidad de explorar soluciones óptimas en un amplio espectro de posibilidades.



Descripción y análisis de las pruebas realizadas:

Para las diferentes pruebas realizadas se estimó un crecimiento de la complejidad mucho mayor a medida de que iba incrementando el número de equipos, pero los resultados sobrepasaron esas estimaciones, el crecimiento esperado a medida de que iba aumentando el valor fue demasiado alto y en cuanto a la generación del calendario con menor costo no se pudo obtener el mejor calendario de acuerdo a esto pero si genera los calendarios que cumplen todas las restricciones.

Conclusiones y aspectos por mejorar:

Desarrollar este proyecto nos hizo resaltar la importancia de la complejidad que puede llegar a tener un algoritmo, el programa parecía tener una complejidad alta para llevarse a cabo pero en realidad su dificultad radica en la eficiencia del código, como podía ejecutarse y arrojar un resultado validó en tiempos óptimos con un número de equipos alto.

La idea principal era generar la mayor cantidad de resultados válidos para escoger el mejor, esto por medio de permutaciones aleatorias entre sus datos y validando las restricciones, pero llevar a cabo esto fue totalmente un reto dado que la complejidad incrementa demasiado con cada validación y según la entrada de los datos. Además, se tuvo que utilizar el concepto de semilla para poder generar un calendario más predecible a la horas de hacer las permutaciones y reducir un poco esa complejidad que nos daba la aleatoriedad.

También nos damos cuenta de la importancia de analizar cada detalle del problema, para generar las permutaciones las podíamos generar de muchas maneras con estructuras de datos diferentes las cuales resultaban en una solución factible, pero a la hora de aplicar las restricciones ocasionaba que su complejidad aumentara en gran medida.

En conclusión, hemos llegado a comprender la importancia fundamental de la optimización y la previsión en el diseño de una estructura flexible, especialmente cuando se enfrentan problemas susceptibles de incrementar significativamente la cantidad de datos a validar. Este reconocimiento destaca la necesidad de anticiparse a desafíos potenciales y de construir sistemas que puedan adaptarse eficientemente a condiciones cambiantes, garantizando así una gestión eficaz ante la expansión de datos en el proceso de validación.