

Instituto Tecnológico de Costa Rica

Campus San Carlos

Escuela de ingeniería en Computación

Análisis de Algoritmos

Segundo Proyecto Programado

Profesor:

Ana Lorena Valerio Solís

Alumnos:

Johan Efrén Rodríguez Salas

Lamberth Aarón González Araya

I Semestre 2023

Tabla de contenidos

[Introducción 3](#_Toc136660222)

[Análisis del problema 3](#_Toc136660223)

[Solución del problema 3](#_Toc136660224)

[Análisis de resultados 3](#_Toc136660225)

[Conclusiones 3](#_Toc136660226)

[Recomendaciones 3](#_Toc136660227)

[Bitácoras y minutas 3](#_Toc136660228)

[Referencias 3](#_Toc136660229)

# Introducción

El presente proyecto tiene como objetivo desarrollar un programa en el lenguaje Java que utilice las ventajas de la programación orientada a objetos (POO) para abordar el problema de asignación de cursos a profesores en una universidad. El propósito principal es implementar tres estrategias de diseño diferentes (algoritmos genéticos, programación dinámica y backtracking) para resolver dicho problema y determinar la eficiencia de cada estrategia mediante mediciones empíricas y analíticas.

A lo largo de este proyecto se define el tipo de datos de prueba, como se implementa cada estrategia, que tipos de estructuras son utilizadas, cuáles son las medidas empíricas resultantes por cada estrategia de programación y los respectivos resultados correspondientes a cada estrategia, dando como conclusión los pros y contras de cada estrategia a la hora de solucionar la tarea designada en el proyecto.

# Análisis del problema

El problema planteado consiste en la asignación óptima de cursos a profesores en una universidad, considerando los datos históricos de calificación de cada profesor en los cursos disponibles. El objetivo es encontrar la mejor asignación que maximice la calidad de la educación y los servicios ofrecidos a los estudiantes. Para lograr esto, se deben tener en cuenta las siguientes consideraciones y restricciones.

En primer lugar, se debe asegurar que la carga de trabajo de cada profesor no exceda un tiempo semanal determinado, el cual equivale a la capacidad de impartir un máximo de 4 cursos. Esto implica que la asignación de cursos debe distribuirse de manera equitativa entre los profesores para evitar la sobrecarga de trabajo.

En segundo lugar, es necesario garantizar que todos los cursos disponibles sean asignados una única vez. Esto significa que no puede haber cursos sin asignar, ya que eso afectaría negativamente la calidad de la educación ofrecida por la universidad.

Por último, es importante tener en cuenta la posibilidad de que algunos profesores queden sin asignación. Esto puede ocurrir si la cantidad de profesores es mayor que la cantidad de cursos disponibles. En tal caso, se debe encontrar la asignación óptima que maximice la calidad de la educación dentro de las limitaciones existentes.

En resumen, el problema de asignación de cursos a profesores presenta restricciones cruciales que deben tenerse en cuenta durante el diseño de las estrategias. Estas restricciones incluyen la limitación de carga de trabajo semanal de los profesores, la asignación única de todos los cursos y la posibilidad de que algunos profesores queden sin asignación. Las estrategias de diseño deben abordar estas restricciones para lograr una solución óptima y eficiente que maximice la calidad educativa ofrecida por la universidad.

# Solución del problema

A continuación, se definen todas las etapas realizadas durante el proyecto para la solución del problema previamente analizado, empleando las distintas estrategias de programación y el manejo de estas.

## Datos utilizados

Para abordar el problema de asignación de cursos a profesores, se utiliza una representación de los datos generados aleatoriamente en formato JSON. Estos datos incluyen el nombre del profesor, la calificación del profesor en cada curso, la cantidad total de cursos y la población deseada para esos datos, según las especificaciones del proyecto. Cabe destacar que las calificaciones son valores enteros desde 0 hasta 10, y para determinar que un profesor no es elegible para un curso determinado se le otorga una calificación de -1 en el mismo.

Los datos generados se almacenan en archivos JSON para facilitar su manejo y acceso desde el programa Java. Se crearon un total de seis archivos de datos, correspondientes a diferentes tamaños y poblaciones, siguiendo las siguientes especificaciones del proyecto:

* Tamaño 3p, 9c: Se generó un archivo llamado "data0.json" que contiene información sobre tres profesores y nueve cursos.
* Tamaño 5p, 15c: Se generó un archivo llamado "data1.json" que contiene información sobre cinco profesores y quince cursos.
* Tamaño 6p, 18c: Se generó un archivo llamado "data2.json" que contiene información sobre seis profesores y dieciocho cursos.
* Tamaño 9p, 27c: Se generó un archivo llamado "data3.json" que contiene información sobre nueve profesores y veintisiete cursos.
* Tamaño 10p, 20c: Se generó un archivo llamado "data4.json" que contiene información sobre diez profesores y veinte cursos.
* Tamaño 10p, 30c: Se generó un archivo llamado "data5.json" que contiene información sobre diez profesores y treinta cursos.

Estos archivos se encuentran ubicados en la carpeta "data" en la raíz del proyecto. Se decidió utilizar un script de Python para generar los datos, debido a su facilidad de uso y agilidad a la hora de la creación de datos de manera aleatoria. Al guardar los datos en formato JSON, se logra una estructura de datos legible y fácil de manipular desde el programa Java.

Debido a que los datos se generaron de manera aleatoria, se hicieron modificaciones posteriores para definir datos de control, los cuales no cumplen con las restricciones del proyecto, de forma que se pueda manejar errores en etapas posteriores del programa. La mayoría de estas modificaciones fueron en “data0.json” debido a que es el archivo menos extenso y más sencillo de manejar a la hora de hacer comprobaciones. Los datos de control van desde definir un conjunto de datos invalido para el cálculo, hasta abordar los puntos débiles de cada estrategia de diseño, los cuales posteriormente se menciona como se resolvieron de manera detallada.

Por último, en cuanto al manejo de estos archivos en el programa Java respecta. El archivo **LoadJson.java** se encarga de cargar archivos JSON y convertirlos en objetos de tipo Alelo para el manejo de los datos. Utiliza la biblioteca **json.simple** para analizar y extraer la información necesaria del archivo. Los objetos de tipo Alelo se almacenan en una lista llamada **alleles**, mientras que los valores del total de cursos y la población se obtienen y almacenan en variables correspondientes en un objeto de tipo **ParsedData**. En resumen, **LoadJson.java** permite cargar y convertir los datos de los archivos JSON para su posterior uso en la solución del problema de asignación de cursos a profesores.

## Estructuras utilizadas

### ParsedData

La clase ParsedData es una estructura de datos utilizada para almacenar los resultados del análisis de los archivos JSON. Tiene los siguientes atributos:

* alleles (ArrayList<Allele>): Una lista de objetos Allele que representan la información de las calificaciones de los profesores en los cursos.
* courses (int): El número total de cursos disponibles.
* population (int): La cantidad de población, es decir, la cantidad de individuos en el algoritmo genético que se utilizará para resolver el problema.

La clase ParsedData también tiene un método toString() sobrescrito para proporcionar una representación en forma de cadena de la instancia de ParsedData, que muestra los valores de los atributos alleles, courses y population.

El diagrama de la estructura ParsedData se muestra a continuación:

### Allele

La clase Allele representa un alelo en el algoritmo genético. Un alelo es una combinación de un profesor, un curso y una calificación.

La clase tiene los siguientes atributos privados:

* professor (Professor): El objeto Professor que representa al profesor asociado al alelo.
* course (Course): El objeto Course que representa al curso asociado al alelo.
* grade (int): La calificación del profesor para el curso.

La clase Allele tiene un constructor que recibe un profesor, un curso y una calificación para inicializar los atributos.

Además, la clase Allele proporciona métodos para acceder y modificar los atributos, como getProfessor(), setProfessor(), getCourse(), setCourse(), getGrade(), setGrade(), así como métodos getAllele() y setAllele() que permiten obtener y establecer el alelo completo.

La clase también tiene un método toString() que devuelve una representación en forma de cadena del alelo, mostrando el nombre del profesor, el nombre del curso y la calificación.

El diagrama de la estructura Allele se muestra a continuación:

### Chromosome

La clase Chromosome representa un cromosoma en el algoritmo genético.

La clase tiene los siguientes atributos privados:

* alleles (ArrayList<Allele>): Una lista de objetos Allele que representan los alelos del cromosoma.

La clase Chromosome tiene varios constructores, uno que recibe un ArrayList<Allele> y otro que recibe una List<Allele>, ambos utilizados para inicializar los alelos del cromosoma.

La clase proporciona métodos para generar un cromosoma aleatorio (generateRandom()), verificar si un alelo es óptimo para la mutación (isOptimalAllele()), verificar si un alelo es válido para un curso (isValidCourse()), verificar si un alelo es válido para un profesor (isValidProfessor()), calcular la función de aptitud (fitness()), obtener y establecer los alelos del cromosoma (getAlleles() y setAlleles()), y un método toString() que devuelve una representación en forma de cadena del cromosoma.

El diagrama de la estructura Chromosome se muestra a continuación:

* Indique en que estructura de datos almacena los datos quemados, que datos tiene, cuales datos definió de control para cumplir con las restricciones del problema.
* Hacer un diagrama con las estructuras utilizadas, listas, pilas, arreglos u otras estructuras, por estrategia de diseño.
* Describa cuál es el cruce realizado.
* Realice un diagrama de flujo o seudocódigo donde explique la lógica que desarrollo, para aplicar cada uno de los algoritmos.
* Describa cuál es el tipo de mutación que se aplicó.
* Describa la función aptitud, cálculo matemático
* Describa la estrategia de la programación dinámica, si los datos se ordenan por curso o profesor de forma descendente o ascendente, que criterio se utiliza para la selección o asignación de cada curso en cada etapa de avance…
* Describa la estrategia del backtracking, como va construyendo la respuesta en cada llamada recursiva, como descarta las posibilidades que no cumplen con las restricciones del problema. Para que tamaños de problemas es posible dar respuesta.

# Análisis de resultados

* Resultados finales, indique que partes están completas, cuales defectuosos, y cuáles no se realizaron y el porqué.
* Adjunte las tablas de todas las mediciones realizadas a sus algoritmos.
* Cálculos realizados.
* Clasificación en notación O grande.
* Gráfico y su respectivo análisis donde compare los comportamientos de los algoritmos.
* Análisis de las tablas de mediciones (empíricas y analíticas).

# Conclusiones

* Según la medición realizada indique cuál estrategia es más eficiente para resolver este problema, justifique su aseveración.
* Responda la siguiente pregunta ¿Conforme crece la talla cuál algoritmo se va haciendo más eficiente?

# Recomendaciones

* Con respecto al alcance del proyecto.

# Bitácoras y minutas

# Referencias

GeeksforGeeks. (2023b). Backtracking Algorithms. *GeeksforGeeks*. <https://www.geeksforgeeks.org/backtracking-algorithms/>

GeeksforGeeks. (2023). Dynamic Programming. *GeeksforGeeks*. <https://www.geeksforgeeks.org/dynamic-programming/>

GeeksforGeeks. (2023a). Genetic Algorithms. *GeeksforGeeks*. <https://www.geeksforgeeks.org/genetic-algorithms/>

OpenAI. (2023, May). ChatGPT Conversation.