• PROGRAMA DE ASIGNATURA

<table>
<tr><td>Nombre</td><td colspan="3">Computación II</td></tr>
<tr><td>Carrera</td><td colspan="3">Ingeniería Matemática</td></tr>
<tr><td>Código</td><td colspan="3"></td></tr>
<tr><td>Créditos SCT-Chile</td><td>Nº Sct</td><td colspan="2">Tbjo. Directo: 6 hrs. pedag. – Tbjo. Autónomo: 9 hrs. cronolog.</td></tr>
<tr><td>Nivel</td><td colspan="3">Expresado en nº de semestre. Si es anual, indicar entre paréntesis</td></tr>
<tr><td>Requisitos</td><td colspan="3">Computación I</td></tr>
<tr><td>Categoría</td><td colspan="3">Obligatorio</td></tr>
<tr><td>Área de conocimiento según OCDE</td><td colspan="3">Ciencias Naturales</td></tr>
<tr><td>Descripción</td><td colspan="3">Contribución al Perfil de Egreso
Proporcionar una base sólida en el análisis y diseño de algoritmos eficientes, fundamentales para la resolución de problemas complejos en diversas disciplinas. A través de la enseñanza de técnicas avanzadas como el análisis de tiempo de ejecución, programación dinámica, algoritmos voraces y en grafos, el estudiante desarrolla habilidades críticas en la modelización y resolución de problemas aplicados, tanto en matemáticas como en física e ingeniería. Estas competencias son esenciales para abordar desafíos en áreas como optimización, computación, y fenómenos de transporte, fortaleciendo la capacidad del profesional para aplicar sus conocimientos matemáticos y de ciencias de la ingeniería de manera efectiva y estratégica.</td></tr>
<tr><td></td><td colspan="3">Resultado de aprendizaje general

Conocer las técnicas básicas para el diseño y análisis de algoritmos computacionales, permitiendo la aplicación de estas en la creación e implementación de algoritmos eficientes para la resolución de problemas computacionales nuevos.</td></tr>
<tr><td></td><td>Resultados de aprendizaje específicos</td><td colspan="2">Unidades temáticas</td></tr>
<tr><td></td><td>Analizar rigurosamente el tiempo de ejecución y la correctitud de algoritmos, utilizando notación asintótica y pruebas formales para evaluar su eficiencia y funcionamiento.</td><td colspan="2">1. Técnicas de análisis de algoritmos:

1.1.- Introducción: Herramientas fundamentales para analizar la eficiencia y la correctitud de algoritmos. Se introduce la notación asintótica (O, Ω, Θ) como medio para describir el comportamiento de los algoritmos en distintos casos, y se revisan estrategias para comparar funciones de crecimiento.

1.2.- Técnicas para analizar algoritmos iterativos y recursivos, incluyendo el uso de árboles de recursión.

1.3.- Para correctitud, se estudian pruebas por inducción, el uso de invariantes de ciclo, y métodos para demostrar la terminación y la validez total y parcial de los algoritmos.</td></tr>
</table>

	Implementar y justificar algoritmos basados en la estrategia de Dividir y Conquistar, identificando cuándo su uso es adecuado y evaluando su complejidad temporal.	**2- Divide y Vencerás**: Dedicada al estudio del paradigma Divide y Vencerás, una estrategia que consiste en dividir un problema en subproblemas más pequeños, resolverlos recursivamente y combinar sus soluciones. 2.1.- Se analiza la eficiencia de algoritmos basados utilizando el Teorema Maestro. 2.2.- Se exploran algoritmos como la búsqueda binaria, el algoritmo de Karatsuba para multiplicación de enteros, el ordenamiento por mezcla (Mergesort) y el algoritmo de Strassen para multiplicación de matrices. 2.3.- Se discuten principios de diseño, casos de uso ideales y análisis de eficiencia.
	Diseñar soluciones óptimas mediante Programación Dinámica, reconociendo subproblemas recurrentes, principios de optimalidad y utilizando tablas o memoización para mejorar la eficiencia.	**3- Programación Dinámica**: La programación dinámica es un paradigma para resolver problemas que exhiben subestructura óptima y sobreposición de subproblemas. En esta unidad se estudian los principios fundamentales de este enfoque, destacando la diferencia entre soluciones basadas en tablas de decisión (bottom-up) y aquellas con memoización (top-down). 3.1.- Se desarrollan e implementan soluciones para problemas clásicos como la sucesión de Fibonacci, el problema de la mochila 0-1, el problema del cambio de monedas, la subcadena común más larga (LCS), y caminos mínimos en matrices. 3.2.- Análisis de la eficiencia temporal y espacial de los algoritmos vistos en la unidad, y se discute cómo identificar cuándo es apropiado aplicar esta técnica.

	Uso de grafos como herramienta para modelar y resolver problemas computacionales.	**4- Algoritmos en grafos**: Introducción al estudio de grafos como estructuras fundamentales para modelar relaciones entre objetos. Se presentan diferentes representaciones, como listas de adyacencia y matrices de adyacencia. A lo largo de la unidad, se enfatiza el uso de grafos como herramienta para modelar y resolver problemas computacionales. 4.1.- Algoritmos básicos de recorrido, como la búsqueda en anchura (BFS) y la búsqueda en profundidad (DFS), y sus aplicaciones en detección de ciclos, análisis de conectividad y ordenamiento topológico. 4.2.- Desarrollo de algoritmos para encontrar caminos más cortos, como el algoritmo de Dijkstra, y para obtener árboles generadores mínimos, como los algoritmos de Prim y Kruskal.
	Desarrollar algoritmos voraces (greedy) y analizar en qué condiciones producen soluciones óptimas, comparando sus ventajas y limitaciones frente a otras estrategias.	**5- Algoritmos voraces**: Se estudia el paradigma voraz (greedy), donde las decisiones se toman siguiendo un criterio local óptimo en cada paso, con el objetivo de construir una solución global eficiente. Se contrastan sus condiciones de aplicabilidad con la programación dinámica, y se presentan criterios para garantizar la correctitud, como el principio de optimalidad y la prueba por intercambio. 5.1.- Problemas clásicos como la selección de actividades, la codificación de Huffman y el problema del cambio de monedas con denominaciones especiales. 5.2.- Estudio de algoritmos voraces aplicados a grafos, como los algoritmos de Prim y Kruskal para encontrar árboles generadores de costo mínimo.

	Distinguir entre problemas tratables e intratables, comprendiendo las clases de complejidad P y NP, el concepto de NP-completitud y su relevancia en el diseño de algoritmos	**6- Complejidad P vs NP**: Esta unidad está dedicada a los fundamentos de la teoría de la complejidad computacional. 6.1.- Introducción a las clases de problemas P, NP, NP-completo y NP-hard, y se presenta el problema abierto P versus NP. 6.2.- Conceptos de reducción polinomial como técnica para establecer la dificultad relativa entre problemas, y se analizan problemas clásicos NP-completos como 3-SAT, la coloración de grafos, el camino Hamiltoniano y el problema del subconjunto suma. 6.3.- Exploración de algoritmos de aproximación como una estrategia práctica para enfrentar problemas intratables, evaluando sus garantías de desempeño y aplicando estos métodos a problemas como la cobertura de vértices y la mochila fraccionaria.
	Metodologías de enseñanza y de aprendizaje La metodología contempla clases expositivas, resolución plenaria de problemas, y aplicación guías de aprendizaje, lo cual se trabajará de forma individual o colaborativa. El trabajo autónomo se desarrollará en base a tareas e implementaciones de algoritmos, las cuales pueden ser resueltas de forma individual o grupal.	
	Procedimientos de evaluación Los procedimientos de evaluación del curso contemplan evaluaciones diagnósticas, formativas y sumativas, alineadas con el contenido de las unidades y los resultados de aprendizaje definidos. Las actividades evaluativas concretas para realizar durante el curso serán claramente definidas al inicio de este.	
	Bibliografía básica **Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C.** (2022). *Introduction to Algorithms* (4th ed.). MIT Press.	