Programación y Algoritmos II (Maestría)

Estructuras de Datos y Algoritmos II (Licenciatura)

Presentación de la asignatura

Profesores

Carlos Segura González (carlos.segura@cimat.mx)

Jean-Bernard Hayet (jbhayet@cimat.mx)

Presentación de la asignatura

- Profesores
  - Carlos Segura González (carlos.segura@cimat.mx)
  - Jean-Bernard Hayet (jbhayet@cimat.mx)
- Página Web (Compartición de material, entrega de tareas)
  - http://moodle.cimat.mx:8080/moodle/
- Breve Descripción
  - Estructuras de datos avanzadas
  - Metodologías para resolución de problemas
  - Algoritmos de amplio uso: prog. competitiva principalmente, big data, BBDD, etc.
  - Diseño de algoritmos y aplicaciones.
  - Demostraciones
- Lunes (08:00 – 09:20), Jueves (08:00 – 09:20) ¿Cambiar?
- Algunas clases de repaso si fuera necesario (nos ponemos de acuerdo).
- Requisitos
  - Fundamentos de programación y lenguaje c++ (STL)
  - Listas, pilas, colas, árboles, algoritmos básicos sobre grafos, complejidad, aspectos básica de DP y Divide y Vencerás.

Metaheuristicas – Presentacion de la asignatura

- Temario
  - Introducción: repaso
  - Estructuras de datos avanzadas
  - Paradigmas de resolución de problemas
  - Grafos
  - Resolución de problemas matemáticos: conteo, big numbers, teoría de juegos
  - Procesamiento de cadenas
  - Geometría computacional

Prog. Avanzada – Presentacion de la asignatura

- Evaluación (se refiere a la materia al completo)
  - Exámenes parciales (probablemente 4): 40 %
    - Diseño e implementación de un algoritmo para la resolución de uno o varios problemas.
    - Algunas preguntas de teoría
    - Modalidad (en mi parte): probablemente combine parte escrito con parte oral.
    - No se puede llevar material, salvo que se diga lo contrario.
  - Tareas: 50%
    - Semanalmente o cada 2 semanas deberán completar una tarea práctica de programación en c++
    - Se indicará una fecha tope de entrega.
    - En todas las prácticas además del código deberán incluir un informe.
    - Si no son capaces de resolverlo, igualmente subir informe y códigos con lo que han probado.
    - Por cada día tarde se baja 0.3 puntos. Después de 10 días de la fecha, ya **no se podrá entregar**.
  - Resolución de problemas UVA/Uhunt: 10%
    - Resolver un total de 40 problemas de los que aparecen en Uhunt. De cada subcategoría que aparece en Uhunt (eligiendo la tercera o cuarta edición del libro) pueden elegir como máximo 3. Al menos 3 problemas de cada tema.
    - Los problemas resueltos tienen que ser de nivel 3 o superior
    - Los problemas que resuelvan no pueden ser los mismos (o muy similares) que los que se han resuelto en las tareas o similares entre ellos. Si tienen duda sobre si es muy similar, pregunten.
    - No hace falta informe, pero comentar código al inicio con pequeña descripción.
    - Fecha de entrega: 23 de mayo (No hay posibilidad de entrega tardía)

Objetivos

- Conocimiento avanzados de estructuras de datos y algoritmos, no exclusivamente orientado a concursos.
- Técnicas de demostración.
- Programar de forma eficiente y rápida algoritmos.
- Mejorar el rendimiento en concursos tipo ICPC, Codeforces, etc.
- Aumentar las probabilidades de éxito en entrevistas de trabajo.

Bibliografía

- Competitive Programming. Steve Halim, Felix Halim.
- Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
- Tutoriales de CodeForces y TopCoder.
- C++ Annotations.

Plataformas de programación – UVA online/Uhunt

- UVA Online, Uhunt
  - UVA Online (https://uva.onlinejudge.org) es un repositorio de problemas de programación gestionado por la Universidad de Valladolid
  - Es la plataforma que van a usar para los problemas finales.
  - Uno de los inconvenientes de UVA Online es que los problemas no están categorizados por tipo ni dificultad.
  - La plataforma Uhunt (http://uhunt.felix-halim.net) de Felix Halim (uno de los escritores de Competitive Programming) mantiene una clasificación de los problemas de UVA tanto por categoría como por nivel de dificultad. Además, te muestra una lista de problemas "pendientes" que se puede ordenar en función de diferentes métricas.

Plataformas de programación – HackerRank

- La usaremos en varias tareas, ya que tiene la posibilidad de subir nuevos problemas.
- Les enviaré un enlace y ahí encontrarán los problemas.
- Sugerencia: tratar de resolver algún problema en HackerRank para tratar de familiarizarse.
- Nota: **la entrega de tareas no es por HackerRank.** Tendrán el problema en HackerRank subido para que puedan orientarse y saber si su solución parece estar bien y es suficientemente rápida, pero deben subir informe y código al Moodle.

Plataforma de programación - Otras

- CodeForces
- TopCoder
- CodeChef
- USACO
- SPOJ
- Project Euler
- Concursos:
  - Google Code Jam
  - Facebook Hacker Cup
  - TopCoder TCO
  - Tuenti Contest
  - ICPC

Primeros pasos

- Darse de alta en UVA online y HackerRank
- Resolver un problema fácil en cada uno de ellos.
- Darse de alta de la web del curso:
  - Si tienen cuenta en el Moodle: buscar el curso “Programación y Algoritmos II (2021)” y usar la contraseña GJR987
  - Si no tienen cuenta, enviarme un correo para darles de alta.
- Una vez que les hayamos creado usuario en la web de la asignatura, entrar, subir una foto a su cuenta, y habrá una “tarea” en la que deben subir el nombre de la cuenta que crearon en UVA online.

Forma de trabajo

- Nos vamos a centrar en problemas acotados, en los que tengamos que aplicar un algoritmo, o máximo una mezcla de unos pocos algoritmos conocidos.
- En algunos casos se les pedirá directamente que implemente un determinado algoritmo, pero en otros casos, la tarea más complicada será identificar cuál es el algoritmo que tienen que usar para resolver el problema, con lo que un objetivo principal es que sepan identificar el tipo de algoritmo a utilizar.
- No nos centraremos en aplicaciones prácticas que usan estos algoritmos internamente. Algunas áreas donde se usan:
  - Optimización
  - Visión computacional
  - Big Data

Recomendaciones – Identificar tipo de problema

- Practiquen en identificar qué tipo de estrategia se puede aplicar para resolver el problema:
  - Búsqueda completa
  - Divide y vencerás
  - Greedy
  - Programación dinámica
  - Etc.
- Un fallo que podemos cometer es solo trabajar con problemas que ya están categorizados
- Buena práctica: escoger varios problemas de distintas áreas que nos interese fortalecer, y mezclarlos todos para no saber de qué tipo es.
- No dedicar demasiado tiempo en la resolución de un problema (normalmente me pongo como límite 1 hora). Después es mejor dejarlo y volver a él más tarde o que alguien mire la editorial y nos de pistas. Esto no aplica a los problemas de la clase: deben resolverlo sin buscar ayuda.
- Es posible que estando en un tema, se ponga un ejercicio de otro tema, así que hay que mantener la mente abierta y no pensar exclusivamente en el tema que estamos dando (aplica también a los exámenes).

Recomendaciones – Fijarse en las restricciones

- Dado un problema y una propuesta de algoritmo, estimen con cuidado el número de pasos necesarios.
- Como aproximación, podemos pensar que por cada segundo de cómputo podemos hacer unos 10^8 pasos (algo más si el acceso a memoria es continuo).
- Traten de identificar y programar el algoritmo más simple que cumpla con cierta seguridad las restricciones. Por ejemplo, si disponemos de 3 segundos y estimamos que en el peor caso nuestro algoritmo usará 2,8 segundos, probablemente sea buena idea seguir pensando en el diseño, y sólo si no se nos ocurre nada más, tratar de sacar el problema con ese algoritmo.

Recomendaciones – Domina un lenguaje de programación y domina c/c++

- En general lo más importante no es el lenguaje que usemos, sino el algoritmo que implementemos. Salvo en contadas excepciones, puedes resolver cualquier problema de un concurso con cualquier lenguaje de programación.

- Es bueno dominar c/c++ porque en general ante la misma implementación es más rápido que el resto. Si sólo se nos ocurre un algoritmo cuyo tiempo estimado se acerca mucho al tiempo límite, sería muy recomendable hacerlo en c/c++.

Recomendaciones – Domina herramientas de testeo y debug de código

- Muchos problemas parecen fáciles pero tienen casos extremo que pueden provocar que fallemos en ciertos casos: dedicar un tiempo a los casos extremo, ejecutar con valgrind o alguna otra herramienta, para ver si hay escapes de memoria, escrituras fuera de rangos, etc.
- Crea un fichero con varios casos para probar antes de hacer el envío de nuestro código: si fallamos seremos penalizados en la mayor parte de concursos.
- No estar metiendo los casos de prueba a mano cada vez que vayamos a probar: escríbanlo sólo una vez en un fichero.
- Entrena en localización de bugs: valgrind
- Compila con –wall y –D_GLIBCXX_DEBUG

Recomendaciones - Repositorio

- Traten de desarrollar sus códigos de forma bastante genérica para poder reutilizar código para otros problemas.
- Créense un repositorio con los códigos genéricos que vayan desarrollando.

Ejemplo 1

- Dada un matriz Q de M x N enteros, compruebe si existe una submatriz de Q de tamaño A x B, cuya media sea igual a E. La entrada es M, N, A, B y Q, E.
- Tiempo límite 3 segundos.
- Restricciones caso 1:
  - 1 <= M, N <= 50
  - 1 <= A <= M
  - 1 <= B <= N
  - Los números de la matriz están en el rango [-10000, 10000]
- Restricciones caso 2:
  - 1 <= M, N <= 1000
  - 1 <= A <= M
  - 1 <= B <= N
  - Los números de la matriz están en el rango [-10000, 10000]

Ejemplo 2

- En el TSP (Traveling Salesman Problem), calcular la distancia entre dos soluciones candidatas. Las soluciones candidatas pueden tener hasta 100,000 ciudades. La distancia se calcula como el número de aristas que aparece en una solución pero no en la otra. Consideramos que la arista (a, b) es diferente a la arista (b, a).

Ejemplo 3

- Nos dan un arreglo de N números enteros, queremos saber si existe una subarreglo (subsecuencia contigua) que sume X.
  - Restricción 1: N = 50,000 y todos los los números son positivos:[1,10^9] Tiempo 1 segundo.
  - Restricción 2: N = 50,000 y existen números positivos y negativos [-10^9,10^9]. Tiempo 1 segundo.