Programación 3

Instituto de Computación
Facultad de Ingeniería, UdelaR

August 1, 2023

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

El Curso I

- Basado en el libro de Kleinberg & Tardos, “Algorithm Design”, 2006, Boston, Pearson/Addison-Wesley.
- El curso se basa fuertemente en la lectura del libro. Las demás instancias son un apoyo a su seguimiento.
- La dedicación semanal esperada de cada estudiante es de unas **15hs**.
- **Teórico**: Se tratarán y jerarquizarán algunos de los conceptos teóricos centrales. **El teórico no reemplaza la lectura del libro sino que la guía y profundiza en sus explicaciones de ser necesario.**
- **Práctico introductorio**: conceptos básicos del teórico, resolución interactiva de ejercicios introductorios

El Curso II

- **Práctico**: Se trabajará de forma interactiva en la resolución detallada de un ejercicio del práctico semanal.
- **Talleres**: Espacio de trabajo guiado por docentes para la resolución de ejercicios del práctico y consultas en general.
- **Laboratorios**: Habrá laboratorios diseñados para ejercitar la resolución de ejercicios del práctico en C++. **Estos laboratorios son opcionales y no serán evaluados.**

Modalidades I

Cada estudiante puede elegir una de la siguientes modalidades:

Tradicional:

En su modalidad tradicional la evaluación consta de dos parciales de 50 puntos cada uno (presenciales). En esta modalidad se aprueba el curso reuniendo al menos 25 puntos entre los dos parciales, y se exonera el curso reuniendo al menos 60 puntos entre los dos parciales.

Evaluación continua:

Además de dos parciales de 50 puntos, se realizan dos controles (presenciales) de 6 puntos cada uno (12 puntos en total) que se suman a los obtenidos en los parciales para la aprobación o la exoneración del curso.

Los controles son pruebas escritas que tienen como objetivo evaluar que cada estudiante lleve el curso al día.

Aprobación

Se aprueba el curso reuniendo al menos 25 puntos entre controles y parciales, y un mínimo de 4 puntos en los controles.

Modalidades III

Exoneración

Se exonera el curso reuniendo al menos 60 puntos entre controles y parciales, y un mínimo de 6 puntos entre controles.

Por ejemplo, si obtengo 3 puntos en cada control (6 en total), y obtengo 54 puntos entre los dos parciales, se exonera el curso.

Parciales

Dos parciales de 50 puntos cada uno (presenciales).

Emparejamiento estable

- Dos conjuntos de personas, del mismo tamaño, $W = \{w_1 \dots w_n\}$, $M = \{m_1 \dots m_n\}$.
- Cada $w \in W$ tiene una lista de preferencia sobre M.
- Cada $m \in M$ tiene una lista de preferencia sobre W.
- *Emparejamiento*: subconjunto de $M \times W$ donde nadie aparece repetido.
- *Emparejamiento perfecto*: además nadie queda solo.
- *Emparejamiento estable*: además no existe ninguna inestabilidad, es decir, un par (m, w) que se prefieren mutuamente a sus respectivas parejas.

Algoritmo de Gale–Shapley

1 Marcar a todos en $M \cup W$ como libres
2 **while** *Existe* $m \in M$ *libre que no se ha propuesto a toda* $w \in W$ **do**
3 | Sea w la persona de mayor preferencia para m a la que aún no se propuso m
4 | **if** *w está libre* **then**
5 | | emparejar m con w
6 | **else if** *w prefiere m a su actual pareja* m' **then**
7 | | dejar m' libre y emparejar m con w
8 | **else**
9 | | w rechaza m
10 | **end**
11 **end**

Algoritmo de Gale–Shapley

- ¿Termina?
- Si termina, ¿produce un emparejamiento?
- ¿perfecto?
- ¿estable?
- ¿Todo esto es independiente del orden en que se escoja m libre?
- ¿Qué pasa si no existe una solución? ¿siempre existe una?

Objetivos del curso

1. Conocer algoritmos clásicos que constituyen una base para la resolución de problemas en computación y aplicarlos para la resolución de problemas concretos.
2. Dominar técnicas generales de diseño de algoritmos y aplicarlas.
3. Razonar con rigurosidad sobre cualidades como la corrección y complejidad de algoritmos.
4. Analizar rigurosamente problemas algorítmicos para identificar limitaciones de cómputo inherentes a cada problema, reconociendo diferentes clases de complejidad.

Cómo pensar y escribir una demostración

- ¡Escribirlas! No alcanza con “saber por dónde sale”.
- Pedir a alguien que las lea, por ejemplo los compañeros de grupo de monitoreo.
- Tener en cuenta el destinatario y el contexto donde se escribe; no es lo mismo un libro introductorio, que un boceto de solución hecho en clase, que una solución hecha en una evaluación.
- Lecturas sobre este tema:
  - “Fundamentos de Algoritmia”, G. Brassard, P. Bratley.
  - “Mathematics for Computer Science”, Eric Lehman, F. Thomson Leighton, Albert R. Meyer.
- Consideraciones similares aplican a la escritura de algoritmos.

Cómo escribir una demostración

- Anunciar el plan para la demostración (por inducción, por absurdo, etc.).
- Mantener un flujo ordenado de razonamiento, no un salpicón de ideas desordenadas (conviene empezar con un borrador y luego pasar en limpio).
- Escribir frases completas, no cadenas de ecuaciones libradas a interpretación por sí solas.
- No abusar de símbolos matemáticos, $\forall x \exists y : \forall z \ldots$. En general un texto puede ser mucho más claro y no menos riguroso.

Cómo escribir una demostración

- Definir toda la notación que se usa, el significado de las variables, etc. Ej.: Sea n la cantidad de ....
- Cuando nos referimos a un algoritmo, es conveniente numerar o etiquetar los pasos a los que nos vamos a referir en la demostración. Ej.: Si la condición del ciclo en el paso X no se verifica, entonces debemos tener $n \leq m$ y por lo tanto ...
- Dividir demostraciones complejas en resultados auxiliares.
- Terminar diciendo claramente cómo se concluye la tesis a partir de lo que hemos desarrollado. Ej. .... este hecho contradice la hipótesis X, por lo cual concluimos Y.

Cómo pensar una demostración

- Desconfiar de las obviedades.
- Implicación ($A \implies B$): Se puede asumir A y obtener B a partir de una secuencia de pasos lógicos, o probar el contrarecíproco ($\neg B \implies \neg A$).
- Equivalencias (Si y solo si, sii, iff, $A \iff B$): Se puede probar ambas implicaciones o construir una cadena de equivalencias. Nuestras pruebas de corrección a menudo son pruebas de equivalencia.
- Descomposición en casos. Ej.: Si $n = m$ .... Si $n \neq m$, uno de ellos es mayor que el otro; supongamos, sin pérdida de generalidad, que $n > m$. Entonces ...
- Por absurdo: Asumir que la proposición es falsa y arribar mediante un razonamiento lógico a una contradicción.
- Por inducción: Muy práctico para analizar algoritmos recursivos (pero no solo en esos casos).

Referencia disponible en línea

“Mathematics for Computer Science”,

Eric Lehman, F. Thomson Leighton, Albert R. Meyer.

Gracias.