Introducción

Segundo semestre 2022

IIC2283

Prof. Nicolás Van Sint Jan

Outline

Outline

¿ A qué nos dedicamos en ciencia de la computación ?

¿ Qué es lo fundamental que hace un **científico de la computación** ?

Construcción de algoritmos

Por **construcción de algoritmos** nos referiremos a lo siguiente:

- Diseño de **algoritmos eficientes** para resolver distintos tipos de problemas.
- Construcción de **cotas inferiores** para el tiempo (u otro recurso relevante como espacio) necesario para solucionar un problema.
  - Demostración de que hay problemas que no pueden ser resueltos de manera eficiente.
- **Implementación eficiente** de los algoritmos diseñados en un modelo de computación.

Un ejemplo fundamental

Ejemplo 1

¿Es el siguiente número un **primo**?

$$
\begin{aligned}
&594363236250881445679738443300610044271230329506694061456935\\
&493654987499908267837823162990672937913416793547138262131162\\
&027654525159743671145416885026759510967807798396037679273587\\
&887606706633886423937222779033920335019140885692470045389062\\
&224534954730489613866855218857728804741777937870098279279181\\
&986655311360896681010943076506752842990211660721362674656217\\
&27307145254397654228320456281897617140 03
\end{aligned}
$$

Un ejemplo fundamental

Ejemplo 2

¿Y este es un número primo?

353558891186560241507955450443100026350520833329175690931503
069103786522881715354942509633300139654062382965565848079004
916894606652959189926884268184126431229594008685198918035846
913197991779695432431942496533152602502723067446447567099323
381684112716739773229350158146448964972920807078481464615296
810209625573422152235960269021291543895089434650235831013442
738561822921356741321645174778916314842888929374700585515051
109386475029198891877239576602057958608968526693840706937826
756677527943071831254461829545495683663193173990823553874959
948723076933474434681343056155904515699197800571147417216047
378365580431179780735564711796226173566466738734002088710112
928530176215958325840978442593181246168167871422728123653467
119983202310996914387703065418662332020193370974465774445835
96769496819213269733

La complejidad de primalidad

¿Puede un algoritmo verificar si un número es primo revisando sus posibles divisores?

Hagamos unos cálculos simples en pizarra...

Y ahora ejecutemos en Python un algoritmo para verificar si un número es primo (el algoritmo ya está disponible en el repositorio del curso).

¿Cómo funciona este algoritmo?

Test de primalidad aleatorizado

El test de primalidad usado es conocido como **test de Miller-Rabin**, y es el resultado de una combinación de técnicas modernas para el diseño de algoritmos.

- En particular el algoritmo es **aleatorizado:** hay una probabilidad de error de a lo más

$$\frac{1}{2^{100}} \approx 7.9 \times 10^{-31}$$

Algunas consideraciones importantes

Al diseñar un algoritmo debemos considerar el modelo de computación sobre el cual será implementado.

- ¿Qué operaciones podemos realizar en el modelo?

Al analizar la complejidad computacional de un algoritmo también debemos considerar el modelo de computación.

- ¿Qué operaciones consideramos al analizar la complejidad de un algoritmo?

Algunas consideraciones importantes

En general, el análisis de la complejidad de un algoritmo se realiza considerando un tipo particular de entradas.

- El **peor caso** es muy utilizado, pero también podemos considerar el **caso promedio**.

Al estudiar un problema debemos tener en cuenta que cotas inferiores se puede demostrar para su complejidad

- Estas cotas inferiores dependen del **modelo de computación** considerado

Algunas consideraciones importantes

Debemos considerar modelos de computación que representen el funcionamiento de una **arquitectura de computadores**.

- Por ejemplo, debemos considerar acceso directo a los datos y la diferencia de costo entre el uso de memoria principal y secundaria

Vamos a considerar un ejemplo que nos servirán para ilustrar los puntos anteriores: **ordenación**.

InsertionSort

El siguiente es un algoritmo clásico para ordenar una lista L de números enteros (de menor a mayor).

```
InsertionSort(L[1...n]: lista de números enteros)
    for i := 2 to n do
        j := i − 1
        while j ≥ 1 and L[j] > L[j + 1] do
            aux := L[j]
            L[j] := L[j + 1]
            L[j + 1] := aux
            j := j − 1
    return L
```

Análisis de la complejidad de insertion sort

Consideramos la **comparación** como la operación a contar, la cual tiene costo 1. Sea L una lista con n números enteros.

- ¿Cuál es el peor caso del algoritmo?
- ¿Cuantas comparaciones realiza el algoritmo en el peor caso, como una función de n?

¿Qué otras operaciones son realizadas por el algoritmo? ¿Cambia el tiempo de ejecución del algoritmo si las consideramos?

Una versión mejorada de insertion sort

Suponiendo que $L[1 \cdots (i-1)]$ ya ha sido ordenada (de menor a mayor), el paso básico del algoritmo es encontrar la posición donde debería ser ubicado $L[i]$

- Además el algoritmo debe colocar $L[i]$ en esta posición.

Para disminuir el tiempo de ejecución del algoritmo podríamos utilizar búsqueda binaria para encontrar la posición correcta para $L[i]$

Una versión mejorada de insertion sort

Consideramos nuevamente la comparación como la operación a contar.

Dado que búsqueda binaria realiza $O(\log_2(i))$ comparaciones para encontrar la posición correcta para $L[i]$, el algoritmo mejorado es de orden $O(n \cdot \log_2(n))$

- Esto puede ser deducido utilizando lo siguiente:

$$\begin{aligned}
\sum_{i=1}^{n} \log_2(i) &= \log_2\left(\prod_{i=1}^{n} i\right) \\
&= \log_2(n!) \\
&\leq \log_2(n^n) \\
&= n \cdot \log_2(n)
\end{aligned}$$

¿Una versión mejorada de insertion sort?

¿Es más eficiente el algoritmo que utiliza búsqueda binaria?

- ¿Ve algún problema en este algoritmo?

Un problema con este algoritmo: ¿Cómo colocar de manera eficiente $L[i]$ en la posición correcta en $L[1 \ldots (i-1)]$?

- Un algoritmo ingenuo toma tiempo lineal en este paso, por lo que el tiempo total del algoritmo sería $O(n^2)$, el mismo orden que para insertion sort

¿Una versión mejorada de insertion sort?

Dos lecciones importantes

- Una operación puede ser cambiada por otra en un algoritmo, esto debe tenerse en cuenta al momento de analizar su complejidad.
- El uso de estructuras de datos es fundamental para implementar de manera eficiente las operaciones requeridas por un algoritmo.

Outline

El objetivo de este curso

Introducir técnicas tanto para el **diseño** como para el **análisis de la complejidad computacional** de un **algoritmo**.

- Técnicas básicas y avanzadas.

Se dará énfasis a:

- La compresión del modelo computacional sobre el cual se diseña y analiza un algoritmo
- El uso de ejemplos de distintas áreas para mostrar las potencialidades de las técnicas estudiadas

Equipo docente

Nicolás Van Sint Jan
- Profesor
- nicovsj@uc.cl
- Sala 10 Magíster, DCC

Dante Pinto
- Cátedra/Coordinación
- drpinto1@uc.cl

Julian García
- Corrector
- jgarcg@uc.cl

Benjamín Farías
- Corrector
- bffarias@uc.cl

Nicholas Mc-Donnell
- Tareas
- namcdonnell@uc.cl

Contenidos del curso

1. **Introducción**
2. **Análisis de la eficiencia de un algoritmo**
3. **Técnicas fundamentales de diseño de algoritmos**
4. **Transformaciones de dominio**
5. **Algoritmos aleatorizados**
6. **Algoritmos en teoría de números**
7. **Técnicas para demostrar cotas inferiores**
8. **Análisis de la eficiencia de un algoritmo más allá del peor caso**

Metodología

Clases: Lunes y miércoles módulo 5.

Sala B13

Ayudantías: Viernes módulo 5.

Sala BC24

Evaluación

1. Tareas.
2. Interrogaciones.
3. Examen.

Tareas

- Cinco tareas durante el semestre (se cuentan las cuatro mejores para el promedio final).
- Enunciado se publicará un día lunes en la mañana.
- La entrega será 7 días hábiles luego de la fecha de publicación hasta las 23:59.
- Consistirán en un problema a resolver utilizando Python con tal de diseñar algoritmos que utilicen las técnicas aprendidas en el curso.
  - Se pedirá que la solución respete una complejidad algorítmica dada.
  - Entrega mediante repositorio de GitHub.
  - Serán corregidas mediante tests automatizados.

Tareas: atrasos

- Se aceptarán entregas atrasadas, pero se aplicará una penalización a la nota de la tarea dependiendo del atraso.

- Se define d_i como el descuento aplicado a la i-ésima tarea

$$d_i = \begin{cases} 0 & \text{entrega sin atraso,} \\ 0.5 & \text{entrega con menos de 24 horas de atraso,} \\ 1.5 & \text{entrega con más de 24 pero menos de 48 horas de atraso,} \\ 3.0 & \text{entrega con más de 48 pero menos de 72 horas de atraso,} \\ 7.0 & \text{en otro caso.} \end{cases}$$

- Si T_i' es la nota correspondiente a la rúbrica de la i-ésima tarea. Luego la nota T_i de la tarea es

$$T_i = \max\{T_i' - d_i, 1.0\}$$

Tareas: cupón excepcional

- El estudiante además cuenta con un **cupón excepcional** para aplazar la entrega de una tarea.
- Se puede utilizar **sólo una vez durante el semestre**.

  Permite extender el plazo de entrega de una tarea sin necesidad de una justificación debido a motivos personales.
- La extensión permite un atraso de hasta 72 horas sin que se aplique descuento alguno.

Los atrasos se contabilizan independiente de si las fechas posteriores a la entrega original son días hábiles o no.

NO se harán excepciones

Interrogaciones y examen

- Dos interrogaciones y un examen.
- Interrogaciones presenciales a las **18:30 horas** y duración de **2 a 3 horas**.
- El examen será presencial a las **9:00 horas**.

Interrogaciones y examen: problema de fuerza mayor

Problema de **fuerza mayor** = contagio covid, "contacto estrecho", o cualquier enfermedad o problema que impida rendir una evaluación presencial.

Para solo una **interrogación**:

- La nota del examen reemplazará la nota de esa interrogación.
- La inasitencia **NO** necesita ser justificada.
  - Se reemplazará **automáticamente** la peor nota por el examen.

Para el **examen**:

- Justificativo según las reglas de la Escuela de Ingeniería en Pregrado.
- Nota P y el examen se rinde a comienzos del primer semestre.

NO se harán **excepciones**.

Fechas de tareas, interrogaciones y examen

	Publicación enunciado	**Entrega**
Tarea 1	Lunes 29 de agosto	Miércoles 7 de septiembre
Tarea 2	Lunes 12 de septiembre	Jueves 22 de septiembre
Interrogación 1	Viernes 30 de septiembre	
Tarea 3	Lunes 3 de octubre	Jueves 13 de octubre
Tarea 4	Lunes 17 de octubre	Miércoles 26 de octubre
Interrogación 2	Miércoles 2 de noviembre	
Tarea 5	Lunes 7 de noviembre	Miércoles 16 de noviembre
Examen	Viernes 2 de diciembre	

"El profesor no se hará responsable por tope de horarios con interrogaciones o exámenes de cursos que se regulen por la programación académica de la Escuela de Ingeniería."

Evaluación

Si $N_1, \ldots, N_k$ es una lista de k notas:

$$\mathrm{AVG}_n(N_1, \ldots, N_k) \;:=\; \text{promedio aritmético de los } n \text{ valores más altos de } N_1, \ldots, N_k.$$

Promedio **Tareas** (**PT**):

$$\mathbf{PT} \;=\; \mathrm{AVG}_4(\; T_1,\; T_2,\; T_3,\; T_4,\; T_5 \;)$$

Promedio **Interrogaciones** y **Examen** (**PE**):

$$\mathbf{PE} \;=\; \mathrm{AVG}_3(\; I_1,\; I_2,\; E,\; E \;)$$

Evaluación

Promedio **Final** (**PF**):

$$\mathbf{PF} \;=\; \frac{1}{2}\cdot\mathbf{PT} \;+\; \frac{1}{2}\cdot\mathbf{PE}$$

El curso se aprueba si, y solo si, **todas** las siguiente condiciones se cumplen:

- $\mathbf{PT} \;\geq\; 2,95$
- $\mathbf{PE} \;\geq\; 3,95$
- $\mathbf{PF} \;\geq\; 3,95$

La nota final del curso **NF** será

$$\mathbf{NF} = \begin{cases} \mathbf{PF} & \text{si aprueba el curso} \\ \min\{\mathbf{PF}, 3.9\} & \text{si no aprueba el curso} \end{cases}$$

Comunicación digital

- Clases, enunciados, pautas, etc..

  Repositorio GitHub: IIC2283/DAA-2022-2

- Anuncios, formularios, etc..

  Canvas / Diseno y Analisis de Algoritmo

- Google Calendar:

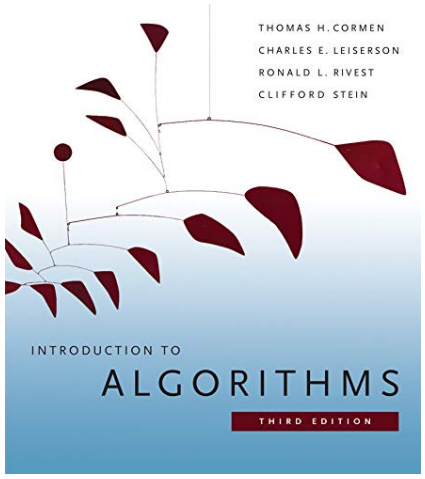
  https://bit.ly/3CIJYbx

- Preguntas sobre materia:
  - Issues de GitHub.
  - Personales: correos de ayudantes / profesor.

- Preguntas por problemas personales relacionados al curso:

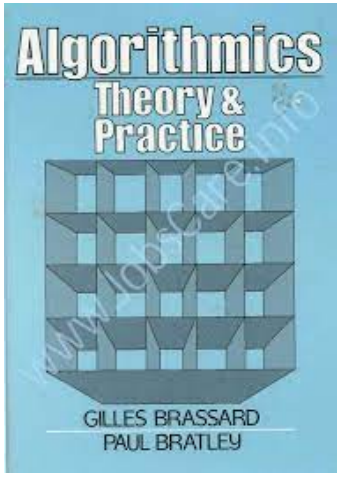
  nicovsj@uc.cl

Bibliografía



Introduction to Algorithms

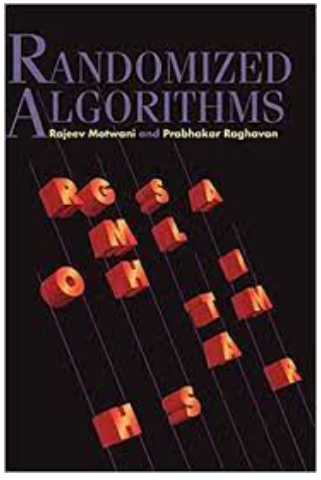
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein.
- MIT Press.
- Tercera edición: 2009.



Algorithmics: Theory and Practice

- Gilles Brassard y Paul Bratley.
- Prentice Hall.
- Primera edición: 1988.

Bibliografía



Randomized Algorithms

- Rajeev Motwani y Prabhakar Raghavan.
- Primera edición: 1995.

Probability and Computing: Randomized Algorithms and Probabilistic Analysis

- Michael Mitzenmacher y Eli Upfal.
- Cambridge University Press. - 2005

Sobre correcciones y recorrecciones

El proceso de **corrección** y **recorrección** de evaluaciones será el siguiente:

1. Entrega de notas y feedback (online): 2 semanas.
2. Recorrección "presencial".
3. Recorrección "escrita".
4. En caso de no quedar satisfecho con recorrección,
   solicitar la recorrección con el profesor (enviar un correo).

En caso de tener cualquier duda sobre corrección o feedback,
enviar un correo a **drpinto1@uc.cl**.

Sobre copia

- Tanto las tareas, interrogaciones y examen son individuales.
- Para las tareas, en caso de querer utilizar una librería en específico preguntar mediante una issue en el repositorio.
- En caso de copia se aplicará:

POLÍTICA DE INTEGRIDAD ACADÉMICA
DEL DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

¿PREGUNTAS?