Estructuras de Datos y Algoritmos – IIC2133

El contenido

Estructuras fundamentales: arreglos, listas ligadas, stacks, colas, tablas de hash, colas priorizadas, listas “skip”

Árboles de búsqueda: árboles binarios, árboles binarios balanceados, otros árboles de búsqueda balanceados

Algoritmos de ordenación: *insertionsort*, *heapsort*, *mergesort*, *quicksort*, análisis de desempeño, ordenación en tiempo lineal

Técnicas algorítmicas: dividir para conquistar, *backtracking*, programación dinámica, algoritmos codiciosos

Grafos: representación, exploración, ordenación topológica, árboles de cobertura mínimos, rutas más cortas, flujo máximo en redes

Las clases

Nos importa que aprovechen las clases, por lo que hemos preparado un sistema de instancias de aprendizaje activo

Podrán reconocer las distintas instancias por sus íconos:

Den ideas

Nos importa que puedan deducir los contenidos del curso

No tengas miedo en decir lo que piensas, nada es obvio

Cuando veas la ampolleta, es el momento de dar ideas

Discute con tus compañeros

Nos importa que puedas discutir ideas con otras personas

¿Cuál crees que es la respuesta y por qué?

Cuando veas los globos de texto, prepara tu argumento

Las tareas

Durante el semestre habrá 5 tareas de programación en **C**

Además, los viernes se harán talleres de **C** con asistencia (AT)

La **nota de tareas** $(\boldsymbol{NT})$ se calcula de la siguiente manera

$$\boldsymbol{NT} = \frac{T_0 + T_1 + T_2 + T_3 + T_4 + AT - \min_{i=0..4}(T_i, AT)}{5}$$

Evaluaciones escritas

Habrán siete controles en clase (los miércoles 20/3, 3/4, 17/4, el lunes 6/5, y los miércoles 23/5, 5/6 y 19/6)

… y un examen (miércoles 26/6)

La **nota de controles** $(\boldsymbol{NC})$ es el promedio de los mejores 6 controles

La **nota de evaluaciones escritas** $(\boldsymbol{NE})$ se calcula como

$$\boldsymbol{NE} = \begin{cases} \dfrac{\boldsymbol{Ex} + \boldsymbol{NC}}{\boldsymbol{2}}, & \boldsymbol{Ex} \geq \boldsymbol{NC} \\ \dfrac{\boldsymbol{Ex} + \boldsymbol{2NC}}{\boldsymbol{3}}, & \boldsymbol{else} \end{cases}$$

La **nota final (NF)** se calcula así

$$\boldsymbol{NF} = \begin{cases} \dfrac{NE + NT}{2}, & si\ NE \geq 4 \\ \min(NE, NT), & en\ otro\ caso \end{cases}$$

Este curso suscribe el **Código de Honor** de la universidad

http://www.uc.cl/codigodehonor/el-codigo

Copias y otros serán sancionados con nota final *NF* = 1.1 en el curso

GitHub: Plataforma oficial del curso

En el repositorio del curso en GitHub podrán encontrar:

- Guías de instalación de C y algunas librerías
- El foro para dudas de tareas, materia, etc.
- Los enunciados de las tareas y las diapositivas de clases
- Sus propios repositorios para entregar las tareas

Deben contestar la encuesta en el **SIDING** para poder acceder

Algoritmos

Nuestra primera ampolleta

¿Qué es un algoritmo?

Algoritmos

Un **algoritmo** y su **implementación** pueden ser distintos

En clases veremos los algoritmos de manera conceptual

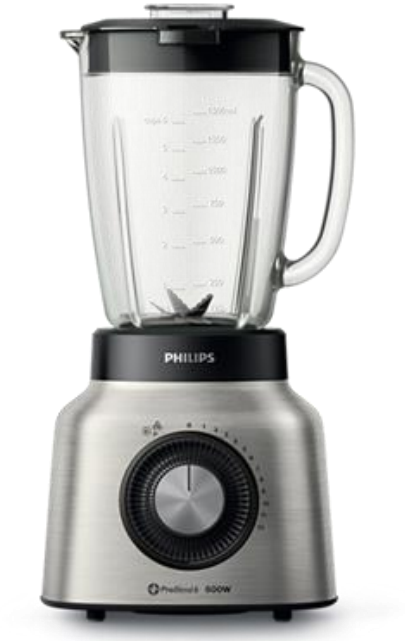
En las tareas tendrán que pensar en la implementación

Como hacer puré de papas

1. Pelar las papas y echarlas en una olla
2. Llenar una olla con agua. Agregar sal a gusto
3. Poner la olla al fuego hasta que las papas estén blandas
4. Sacar las papas del agua y vaciar la olla
5. **Moler** las papas y volverlas a poner en la olla
6. Agregar leche, calentar y revolver. Servir caliente.

Como hacer puré de papas

¿Cómo **moler** las papas?



Todos muelen… pero no de igual manera

Buenos algoritmos

¿Qué hace que una solución a un problema sea buena?

¿Cuál será la mejor?

Complejidad

$f(x) \in O\big(g(x)\big)$:

$$\exists\, x_0, k > 0$$

$$f(x) < k \cdot g(x), \qquad \forall\, x\ > x_0$$

$f(x) \in \Omega\big(g(x)\big)$:

$$g(x) \in O\big(f(x)\big)$$

$f(x) \in \Theta\big(g(x)\big)$:

$$f(x) \in O\big(g(x)\big) \qquad y \qquad f(x) \in \Omega\big(g(x)\big)$$

Complejidad: reglas básicas

- Si un algoritmo tiene varias partes que se ejecutan una después de otra (secuencialmente), su complejidad es la **suma** de las complejidades de cada parte
- Por lo tanto, si una parte se repite x veces (p.ej., un *loop*), entonces (a veces) se puede **multiplicar** su complejidad por x, y luego sumar al resto del algoritmo
- Al final, sólo queda el término que **crece más rápido**

Complejidad de tiempo y memoria

Nos interesan dos tipos de complejidades para algoritmos:

- Complejidad de tiempo: $T(n)$
- Complejidad de memoria: $M(n)$

Ambos son relativos al **tamaño del input** (i.e., el número de datos de entrada), n

Si bien $T(n)$ es nuestra prioridad, nunca olvidar que

$$T(n) \in \Omega\big(M(n)\big)$$

¿Por qué C?

El lenguaje C posee las siguientes características:

- Control absoluto de memoria
- Control absoluto de la ejecución del algoritmo
- No trae nada que no sea necesario

Esto significa que el lenguaje tiene mínimo *overhead*

¿Por qué no **Python**?

El lenguaje Python posee las siguientes características:

- Estructuras de datos integradas
- Fácil y rápido de escribir
- Se encarga de casi todo para comodidad del usuario

Esto significa que el lenguaje muchas veces hace cosas sin que uno sepa, aumentando la complejidad del programa

Taller 0

Este viernes será el primer taller introductorio a **C**.

Necesitan lo siguiente:

- Haber leído y seguido la guía de instalación de **C**
- Haber leído la guía de **arreglos**
- Haber leído la guía de **punteros**
- Traer su computador. Si no tiene, puede trabajar con un compañero

Próxima Clase

La próxima clase comenzaremos con los contenidos del curso

Deben traer leída la guía de **arreglos** que está en el repositorio (aunque no vayan al taller)

Deben tener clara la diferencia entre una lista y un arreglo