# Estructuras de datos

## Sesión 1

David Charte <fdavidcl@ugr.es> https://deivi.ch

# Presentación prácticas

# Clase martes (A3)

https://meet.google.com/ahk-yzfb-ayp

| Turno | Fecha | Contenidos |
|---|---|---|
| 1 | 29/09/20 | Eficiencia |
| 2 | 06/10/20 | Abstracción (TDA Racional) |
| 1 | 13/10/20 | Abstracción (TDA Imagen) |
| 2 | 20/10/20 | Abstracción (TDA Imagen) |
| 1 | 27/10/20 | TDA lineales (intro) |
| 2 | 03/11/20 | TDA lineales (sin STL) |
| 1 | 10/11/20 | TDA lineales (con STL) |
| 2 | 17/11/20 | TDA no lineales (intro) |
| 1 | 24/11/20 | TDA no lineales (dudas) |
| 2 | 01/12/20 | Árboles (binarios) |
| 1 | 08/12/20 | Árboles (APO/ABB/AVL) |
| 2 | 15/12/20 | Práctica final (intro) |
| 1 | 22/12/20 | Práctica final (dudas) |
| 2 | 12/01/20 | Práctica final (dudas) |

- Página docencia y tutorías: https://deivi.ch/docencia y fdavidcl@ugr.es

# Clase viernes (A2)

https://meet.google.com/uhq-tcah-dzn

| Turno | Fecha | Contenidos |
|---|---|---|
| 1 | 02/10/20 | Eficiencia |
| 2 | 09/10/20 | Abstracción (TDA Racional) |
| 1 | 16/10/20 | Abstracción (TDA Imagen) |
| 2 | 23/10/20 | Abstracción (TDA Imagen) |
| 1 | 30/10/20 | TDA lineales (intro) |
| 2 | 06/11/20 | TDA lineales (sin STL) |
| 1 | 13/11/20 | TDA lineales (con STL) |
| 2 | 20/11/20 | TDA no lineales (intro) |
| 1 | 27/11/20 | TDA no lineales (dudas) |
| 2 | 04/12/20 | Árboles (binarios) |
| 1 | 11/12/20 | Árboles (APO/ABB/AVL) |
| 2 | 18/12/20 | Práctica final (intro) |
| 1 | 08/01/20 | Práctica final (dudas) |
| 2 | tutoría colectiva? | Práctica final (dudas) |

- Página docencia y tutorías: https://deivi.ch/docencia y fdavidcl@ugr.es

# Contenidos de las prácticas

# Contenidos de las prácticas

1. Eficiencia: pequeñas tareas para medir eficiencia de forma teórica y práctica
   - voluntaria: puntuación para punto de participación
2. Abstracción: uso de clases y objetos para encapsular datos (0.5 hasta 0.75, parejas)
   - Uso de Makefiles
   - Uso de Doxygen
   - Especificación e implementación de tipos de dato abstractos (Racional, voluntaria)
   - Especificación e implementación de tipos de dato abstractos (Imagen)
3. TDA lineales: pila con máximo (0.5)
4. TDA no lineales: diccionario y guía de teléfonos con STL (0.5)
5. Árboles: uso de árboles binarios, AVL, ABB y APO (voluntaria)
6. Práctica final: implementación de un programa para visualizar rutas aéreas sobre un mapa mundi (1.25, parejas)

# Herramientas para trabajar

# Herramientas

- Compilador: recomendado GCC y GNU Make
- Editor: recomendado VS Code/Code-OSS/Codium (Emacs o Vim si ya los conoces)

# Práctica 1: Eficiencia

# ¿Por qué es importante calcular la eficiencia?

- Datos de juguete vs datos reales
- Importancia de las aproximaciones lineales a problemas complejos

# Diferencia entre complejidad y eficiencia

- La complejidad se asocia al problema: un problema se dice NP si es resoluble por un sistema de búsqueda no determinista en tiempo polinómico, se dice P si es resoluble en tiempo polinómico por un algoritmo de búsqueda determinista
- La eficiencia está ligada al algoritmo: un problema NP que no sea P no tendrá algoritmos que lo resuelvan en $O(n)$, $O(n^2)$, pero un problema P puede tener distintas soluciones. Ejemplo: multiplicación.
  - El algoritmo estándar de multiplicación tiene una eficiencia de alrededor de $O(n^2)$, pero el algoritmo de Karatsuba utiliza $O(n^{log_2(3)}) = O(n^{1.58})$ -> 59K operaciones vs 1M operaciones para dos números de 1024 dígitos.
  - El límite parece que es $O(n \log n)$ https://www.quantamagazine.org/mathematicians-discover-the-perfect-way-to-multiply-20190411/

# Eficiencia teórica

Se mide el número de operaciones (mejor, peor caso o caso promedio) que llevaría aplicar un algoritmo a datos de un cierto tamaño $n$. Las operaciones elementales son:

- operaciones aritmético-lógicas
- indexaciones (accesos)
- asignaciones
- incrementos

Por tanto, se asume que los accesos son O(1), las multiplicaciones también...

## Reglas de cálculo (diapositivas Gustavo)

- Los números son O(1), los coeficientes desaparecen: $2n + 3 = O(n)$

## Ejemplo: búsqueda binaria (diapositiva)

# Eficiencia empírica

Medimos el tiempo real que tarda en ejecutarse un programa con datos de tamaño variable.

Para un mismo algoritmo, muchos factores pueden variar el tiempo:

- Optimizaciones del compilador
- Lenguaje de programación
- Hardware
- ...

Por lo general, la **tendencia** con datos más grandes vendrá dada por la eficiencia teórica. Con tamaños pequeños, estos detalles influyen mucho más y, con tamaños grandes, puede influir como un factor (Python/Ruby pueden trabajar 5-10 veces más lento que C).

# Midiendo tiempos

En C++: usar `<chrono>`: se toma un `time_point` al principio de la ejecución, otro al final y se restan obteniendo un `duration`. Se pueden imprimir por pantalla utilizando `std::chrono::microseconds` y `std::chrono::nanoseconds` para especificar la precisión:

```
std::cout << std::chrono::duration_cast<std::chrono::microseconds>(t2 - t1);
```

# Visualizando tiempos

Herramientas: gnuplot (scripting), ggplot2 (R), matplotlib (Python)

```
import numpy as np
import matplotlib.pyplot as plt
tiempos = np.loadtxt("tiempos.dat", delimiter="\t")
plt.plot(tiempos[:,0], tiempos[:,1])
plt.show()
```

https://matplotlib.org/gallery/lines_bars_and_markers/simple_plot.html

# Compilando código con Makefiles

- Sección principal:
  - declaración de variables `NOMBRE=valor`
  - declaración de objetivos `objetivo: dependencias`
- Dentro de cada objetivo:
  - nombre del objetivo: puede ser de tipo `.PHONY` o falso (no se refiere a un archivo). Se nota como `$@` en la receta.
  - dependencias: son archivos que deben existir y estar actualizados previamente a construir el objetivo. Se notan como `$<` (primera) y `$^` en la receta.
  - receta: secuencia de órdenes de terminal para construir el objetivo, por ejemplo, una línea de compilación. En ocasiones, GCC puede **inferir** la receta por nosotros (si los archivos tienen extensiones estándar).

Makefile mínimo:

```
CPPFLAGS=-O2 -Wall
default: programa
```

GNU Make infiere la línea de compilación: `g++  -O2 -Wall programa.cpp  -o programa`

# Entrega

Entregar **código e informe(s)** (un solo pdf con todos los ejercicios es perfectamente válido).

En cada ejercicio, explicar cuál debería ser la eficiencia teórica, comprobar si coincide con la empírica, si no coincide tratar de explicar por qué.