INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO


# ANÁLISIS Y DISEÑO DE ALGORITMOS

## PRÁCTICA 10

Rodrigo García Mayorga

2012630554

# Introducción

El **problema de la mochila** es un clásico en la teoría de algoritmos y la optimización combinatoria. Consiste en seleccionar un subconjunto de elementos con valores y pesos dados, de manera que se maximice el valor total sin exceder la capacidad de una mochila (o contenedor).

## Descripción Formal

Dado:

- Un conjunto de `n` objetos.
- Cada objeto `i` tiene:
  - un **valor** `v[i]`
  - un **peso** `w[i]`
- Una capacidad máxima `W` de la mochila.

El objetivo es seleccionar un subconjunto de los objetos que:

- **Maximice** la suma de los valores de los objetos seleccionados.
- Sin que la suma de sus pesos **supere** `W`.

## Tipos de Programas para Resolverlo

Hay varios enfoques para resolver este problema, siendo los más comunes:

1. **Algoritmo de Fuerza Bruta**: Prueba todas las combinaciones posibles (ineficiente para `n` grandes).
2. **Programación Dinámica (0/1 Knapsack)**:
   - Ideal cuando los objetos **no se pueden fraccionar**.
   - Tiempo: `O(nW)`
   - Usa una tabla `dp[i][w]` para representar el máximo valor con los primeros `i` objetos y capacidad `w`.
3. **Algoritmo Greedy (para el problema fraccional)**:
   - Se usa cuando los objetos **se pueden dividir**.
   - Ordena los objetos por su relación valor/peso y los agrega mientras haya espacio.

# Desarrollo

## 1. Descripción del Problema

El problema consiste en encontrar la combinación óptima de objetos para maximizar el valor total sin exceder el peso máximo de 4 libras.

## 2. Objetos Disponibles

| Objeto | Peso (lb) | Valor ($) |
|---|---|---|
| Guitarra | 1 | 1500 |
| Laptop | 3 | 2000 |
| Estéreo | 4 | 2000 |
| iPhone | 1 | 2000 |

## 3. Implementación

- **Lenguaje**: C
- **Algoritmo**: Programación Dinámica (Knapsack)
- **Estructura de datos**: Matriz bidimensional para almacenar soluciones parciales

## 4. Componentes Principales del Código

1. Función `max()`: Compara dos valores y devuelve el mayor
2. Función `knapsack()`: Implementa el algoritmo principal
3. Matriz `dp[][]`: Almacena los valores óptimos para cada subproblema

## 5. Complejidad del Algoritmo

- **Tiempo**: O(n*W) donde n es el número de objetos y W es la capacidad máxima
- **Espacio**: O(n*W) para la matriz de programación dinámica

## 6. Compilación y Ejecución

```
gcc mochila.c -o mochila
```

```
./mochila
```

## 7. Resultados Esperados

El programa debe encontrar la combinación óptima que maximice el valor total respetando el límite de peso de 4 libras, mostrando:

- Valor máximo alcanzable
- Lista de objetos seleccionados
- Peso y valor individual de cada objeto seleccionado

## 8. Validación

- El programa maneja correctamente los casos límite
- Los resultados son consistentes con la tabla de valores proporcionada
- La solución respeta las restricciones de peso

## 9. Conclusiones

El programa implementa exitosamente una solución al problema de la mochila mediante programación dinámica, proporcionando resultados óptimos y detallados sobre la selección de objetos.

## Enlace del código (github):

https://github.com/LowisN/Practica-10