Práctica 5. Algoritmos Voraces (Greedy)
GUIÓN DE LA PRÁCTICA

1.- Objetivos de la práctica

- Entender los fundamentos de la estrategia voraz (greedy)
- Implementar un algoritmo voraz sencillo para un problema cotidiano
- Analizar la eficacia de diferentes estrategias voraces
- Practicar la implementación básica en C++

2.- Actividades a realizar

Problema 1: La Ruta de la Tapa

¡Ha llegado el fin de semana de la Ruta de la Tapas a la ciudad! Tu grupo de amigos quiere visitar varios bares para probar sus tapas especiales. Ahora bien, tenéis un tiempo limitado (solo 3 horas) pero queréis maximizar el disfrute de vuestra experiencia gastronómica.
Para mantener el problema accesible, utilizaremos un modelo simplificado:

1. Partiréis de un punto central (la Plaza Mayor de la ciudad)
2. Visitaréis cada bar elegido en orden, regresando al punto central después de cada visita para reagruparos y decidir la siguiente visita.
3. Cada bar tiene las siguientes características:
    - Nombre del bar
    - Valoración de su tapa (de 1 a 10 puntos) basada en las reseñas recibidas
    - Tiempo necesario para consumir (entre 15 y 30 minutos, incluyendo espera y consumo)
    - Tiempo de desplazamiento desde/hacia el punto central (en minutos caminando)

Conjunto de datos

Este año, estos son los bares que participan en la Ruta:

Nombre	Valoración	Tiempo Consumición (min)	Tiempo Desplazamiento (min)
El Rincón del Choco	9	30	10
Bar La Esquina	6	15	5
Tapas & Cañas	4	15	5
Casa Manolo	8	25	5
La Bodeguita	7	20	7
El Mirador	10	30	12
Mesón del Puerto	5	15	5
La Taberna Asturiana	8	20	10
El Rincón del Abuelo	3	25	12
Tapería Central	6	20	7

El algoritmo voraz

El objetivo es seleccionar los bares a visitar para maximizar la suma de valoraciones de las tapas, sin exceder el tiempo disponible.

```
/**
 * Algoritmo voraz para seleccionar los bares a visitar.
 *
 * @param bares Vector de objetos Bar con la información de cada bar
 * @param tiempoDisponible Tiempo máximo disponible en minutos
 * @return Vector de bares seleccionados
 */
vector<Bar> RUTATAPAS(vector<Bar> bares, int tiempoDisponible) {
    // Ordenar bares según cierto criterio
    ordenar(bares.begin(), bares.end(), criterio);

    int tiempoRestante = tiempoDisponible;
    vector<Bar> baresSeleccionados;

    for (const Bar& bar : bares) {
        // Tiempo total = ida + vuelta + consumición
        int tiempoNecesario = (2 * bar.tiempoDesplazamiento) + bar.tiempoConsumicion;

        if (tiempoRestante >= tiempoNecesario) {
            baresSeleccionados.push_back(bar);
            tiempoRestante -= tiempoNecesario;
        }
    }

    return baresSeleccionados;
}
```

Actividades:

1. Implementación

a) Implementa en C++ el algoritmo voraz utilizando el siguiente criterio de selección:

- **Estrategia 1**: Escoger primero los bares con mejor valoración (de mayor a menor)

b) Ejecuta el algoritmo con un tiempo disponible de 180 minutos (3 horas) y muestra (capturas de pantalla):

- Los bares seleccionados
- La valoración total conseguida
- El tiempo total empleado

2. Comparación de estrategias

a) Implementa otras dos estrategias voraces:

- **Estrategia 2**: Escoger primero los bares con menor tiempo total de visita (de menor a mayor)
- **Estrategia 3**: Escoger primero los bares con mejor ratio (valoración/tiempoTotal)

b) Ejecuta las tres estrategias con los mismos datos y compara los resultados. Para cada estrategia, muestra (capturas de pantalla):

- Los bares seleccionados
- La valoración total conseguida
- El tiempo total empleado

c) ¿Cuál de las tres estrategias dio mejor resultado? ¿Por qué crees que ocurre esto?

Ejemplo de salida esperada

Para ayudarte a verificar tu implementación, te proporcionamos un ejemplo de salida esperada para la Estrategia 1 (ordenar por valoración):

```
ESTRATEGIA 1: Ordenar por valoración (de mayor a menor)
--------------------------------------------------
Bares seleccionados:
1. El Mirador (Valoración: 10, Tiempo total: 54 min)
2. El Rincón del Choco (Valoración: 9, Tiempo total: 50 min)
3. Casa Manolo (Valoración: 8, Tiempo total: 35 min)
4. La Taberna Asturiana (Valoración: 8, Tiempo total: 40 min)

Valoración total conseguida: 35 puntos
Tiempo total empleado: 179 minutos
Tiempo restante: 1 minutos
```

Problema 2: Maratón de Series

¡Por fin llega el fin de semana y has encontrado un hueco entre prácticas, AADs y exámenes para ver tus series pendientes! Ahora bien, como no todo va a ser ocio, tu tiempo para esta tarea es limitado: solo dispones de diez horas; eso sí, quieres aprovechar al máximo el tiempo que tienes disponible y ver las series que más te interesan.

Importante: En este atracón televisivo, la unidad indivisible es el capítulo (se considera un crimen dejar un capítulo a medias...); aunque sí podemos elegir ver todos los capítulos de una serie, solo algunos o ninguno.

Datos del problema

Cada serie en tu lista tiene estas características:

- Título de la serie
- Número de episodios que tienes pendientes
- Duración por episodio (en minutos)
- Valoración de interés (de 1 a 10 puntos)
- Género (comedia, drama, acción, ciencia-ficción, fantasía)

Lista de series disponibles

Para esta práctica, trabajaremos con las siguientes 10 series:

Título	Episodios	Duración (min)	Valoración	Género
Stranger Things	3	50	8	ciencia-ficcion
Breaking Bad	2	45	9	drama
The Office	5	22	7	comedia
Game of Thrones	2	60	9	fantasía
Brooklyn Nine-Nine	4	22	6	comedia
The Mandalorian	2	40	8	ciencia-ficcion
Peaky Blinders	3	55	7	drama
The Witcher	2	60	7	fantasía
Money Heist	3	45	8	acción
The Boys	2	55	8	acción

El algoritmo de mochila

El objetivo es seleccionar qué capítulos de cada serie ver para maximizar la suma de valoraciones de interés, sin exceder el tiempo disponible.

```
maratonseries_capitulos(series[1..n], tiempoDisponible):
    // Información de cada serie
    // series[i] = {titulo, episodios, duracion, valoracion, genero}

    // Crear lista de todos los capítulos individuales
    capitulos = []
    para cada serie s en series:
        para cada episodio e de 1 a s.episodios:
            capitulos.añadir({
                serie: s,
                numero_episodio: e,
                duracion: s.duracion,
                valoracion: s.valoracion // Cada capítulo tiene la valoración completa de la serie
            })

    // Ordenar capítulos según cierto criterio
    ordenar(capitulos) según criterio elegido

    tiempoRestante = tiempoDisponible
    capitulosSeleccionados = []

    para cada capitulo c en capitulos:
        si tiempoRestante >= c.duracion:
            añadir c a capitulosSeleccionados
            tiempoRestante = tiempoRestante - c.duracion

    // Agrupar los capítulos seleccionados por serie
    seriesSeleccionadas = agruparPorSerie(capitulosSeleccionados)

    devolver seriesSeleccionadas
```

Actividades

1. Implementación

a. Implementa en C++ el algoritmo de mochila utilizando este criterio de ordenación:
   - **Estrategia 1**: Ordenar capítulos por valoración de la serie (de mayor a menor)

a. b) Ejecuta el algoritmo con un tiempo disponible de 600 minutos (10 horas) y muestra (captura de pantalla):
   - Las series seleccionadas con el número de capítulos vistos de cada una
   - La valoración total conseguida
   - El tiempo total empleado
   - El número total de capítulos que verás

2. Comparación de estrategias

a. Implementa otras dos estrategias voraces:
   - **Estrategia 2**: Ordenar capítulos por duración (de menor a mayor)
   - **Estrategia 3**: Ordenar capítulos por ratio (valoración_por_capítulo/duración), de mayor a menor

b. Ejecuta las tres estrategias con los mismos datos y compara los resultados. Para cada estrategia, muestra:
   - Las series seleccionadas con el número de capítulos vistos de cada una
   - La valoración total conseguida

- El tiempo total empleado
- El número de capítulos por género

c. ¿Cuál de las tres estrategias dio mejor resultado? ¿Por qué crees que ocurre esto?
d. A diferencia de la mochila fraccionaria clásica, ¿crees que la Estrategia 3 es óptima en este caso? Justifica tu respuesta.

3.- Entrega

- La fecha de entrega será la del día correspondiente al turno de práctica (L1-L9) de la **semana del 12 de mayo**.
- La entrega de la práctica se realizará por Moodle en la tarea puesta al efecto y tendrá el siguiente formato:

Crear y subir una carpeta comprimida ***Apellido1Apellido2Nombre.rar*** la cual debe contener:

- Una subcarpeta con las fuentes del programa (.cpp y .h)
- Una subcarpeta con la memoria de la práctica, de 12 páginas como máximo, en formato pdf con el esquema especificado a continuación:
  - Resultados obtenidos con cada estrategia
  - Análisis comparativo de las estrategias
  - Conclusiones obtenidas
  - Respuestas a las preguntas planteadas
- Una subcarpeta con todo el material suplementario que consideres oportuno (tablas de datos, hojas de cálculos, gráficas, etc…)