Práctica 3: *Fuerza Bruta* vs. *Divide y Vencerás*

Programación-II

Dpto. de Informática e Ingeniería de Sistemas,
Grado de Ingeniería Informática
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

1. Objetivo de la práctica

El objetivo principal de la práctica es resolver un problema mediante dos estrategias diferentes: a) *Fuerza Bruta*; b) *Divide y Vencerás*.

2. Problema: Calcular el máximo solape de un conjunto de intervalos

En esta práctica se propone diseñar algoritmos para resolver el problema del máximo solape entre intervalos. Dado un conjunto de intervalos definidos en los reales, calcular el máximo solape consiste en encontrar la pareja de intervalos cuyo solape entre sí es máximo. Por ejemplo, en el conjunto de intervalos $[4.0, 7.5]$, $[3.0, 5.0]$, $[0.5, 2.5]$, $[1.5, 8.0]$, $[0.0, 1.5]$, $[2.0, 4.0]$, $[1.0, 6.0]$, $[3.5, 7.0]$, que se representan gráficamente en la Figura 1, el máximo solape es el de los intervalos $[1.0, 6.0]$ y $[1.5, 8.0]$ que es igual a 4.5.

3. Tareas

El problema del máximo solape se resolverá mediante dos estrategias distintas: a) *Fuerza Bruta*; b) *Divide y Vencerás*.

Se pide realizar las siguientes tareas:

- Diseñar e implementar la estrategia de *Fuerza Bruta* para resolver el problema. El coste asintótico en tiempo será $\mathcal{O}(n^2)$ donde n es el número de intervalos.

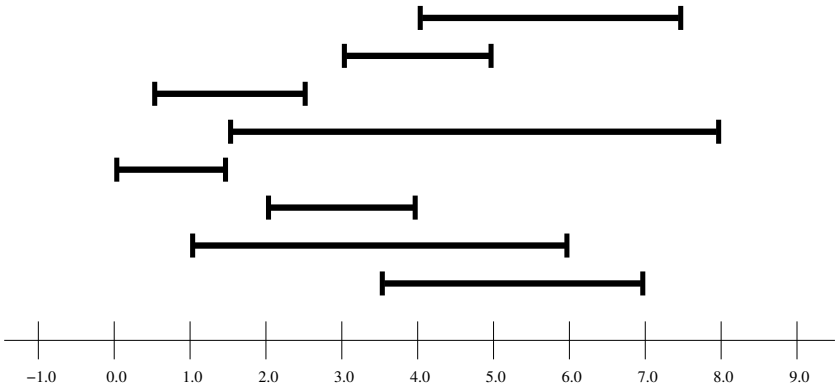

Figura 1: Intervalos definidos en los reales.

- Diseñar e implementar la estrategia de *Divide y Vencerás* para resolver el problema. El coste asintótico en tiempo será $\mathcal{O}(n \cdot log(n))$ donde n es el número de intervalos.
- Implementar un programa que pruebe las estrategias implementadas. Dicho programa se debe guardar en el fichero `calmsolape.cpp`.

4. Código de apoyo

Como código de apoyo, disponible en Moodle, se proporcionan el fichero `maxsolape.hpp` que se muestra a continuación. Aunque se pueden implementar funcionales adicionales, obligatoriamente deberán implementarse la funciones especificadas en `maxsolape.hpp`.

```
const int N = 100000;     // Maximo numero de intervalos
const double minini = 0; // Minimo valor de inicio para los intervalos
const double maxfin = 100; // Maximo valor de fin para los intervalos

struct tpSolape{
    int interA, interB; // Indice de los intervalos
    double solape;     // Solape de los intervalos
};

struct tpInter{
    int ind;           // Indice del intervalo en la matriz inicial
    double ini, fin;  // Inicio y fin del intervalo
};

// Cada fila de inters representa un intervalo. La primera columna
// es el inicio del intervalo, y la segunda el final. Por ejemplo:
//   double inters[N][2] = {
//       {1.5, 8.0},
//       {0.0, 4.5},
//       {2.0, 4.0},
```

```
//       {1.0, 6.0},
//       {3.5, 7.0}
//   };
// tiene cinco intervalos, el primero empieza en 1.5 y termina en 8.0.


// maxSolFBruta devuelve un registro tpSolape en el que el campo solape
// es el maximo solape entre parejas de los n primeros intervalos de inters,
// y los campos interA e interB son los indices de dichos intervalos.
// Para la matriz inters de ejemplo, el resultado es solape=4.5, interA=0,
// interB=3
// (los valores de interA e interB pueden estar intercambiados, es decir,
// el resultado para el ejemplo anterior también puede ser solape=4.5,
// interA=3, interB=0).
tpSolape maxSolFBruta(double inters[N][2], int n);

// Crea un vector de tpInter con los n primeros intervalos de inters.
// Por ejemplo para la matrix inters de la funcion anterior y n=5, los
// valores de indinters seran:
// [{ind: 0, ini: 1.5, fin: 8.0},
//  {ind: 1, ini: 0.0, fin: 4.5},
//  {ind: 2, ini: 2.0, fin: 4.0},
//  {ind: 3, ini: 1.0, fin: 6.0},
//  {ind: 4, ini: 3.5, fin: 7.0}]
void crearvind(double inters[N][2], tpInter indinters[N], int n);

// Ordena con el algoritmo mergesort los intervalos de indinters
// comprendidos entre las componentes indexadas por p y f, ambas incluidas,
// de acuerdo al valor de inicio de los intervalos (orden creciente).
// Por ejemplo, para el vector de la funcion anterior, p=0 y f=4, el vector
// ordenado sera:
// [{ind: 1, ini: 0.0, fin: 4.5},
//  {ind: 3, ini: 1.0, fin: 6.0},
//  {ind: 0, ini: 1.5, fin: 8.0},
//  {ind: 2, ini: 2.0, fin: 4.0},
//  {ind: 4, ini: 3.5, fin: 7.0}]
void mergesortIndInters(tpInter indinters[N], int p, int f);

// Dado un vector indinters, utiliza la tecnica de Divide y Venceras para
// devolver el maximo solape entre parejas de intervalos comprendidos
// entre las componentes indexadas por p y f, ambas incluidas.
// Por ejemplo, para el vector del procedimiento anterior,
// el resultado es solape=4.5, interA=0, interB=3
tpSolape maxSolDyV(tpInter indinters[N], int p, int f);
```

Probablemente necesitarás implementar funciones que no están declaradas en `maxsolape.hpp`. Dichas funciones deben implementarse en `maxsolape.cpp` y sus cabeceras no deben incluirse en `maxsolape.hpp`.

5. Sobre las estrategias a implementar

Fuerza Bruta: La estrategia de *Fuerza Bruta* se basa en comparar todas las parejas de intervalos entre sí y seleccionar la pareja con máximo solape.

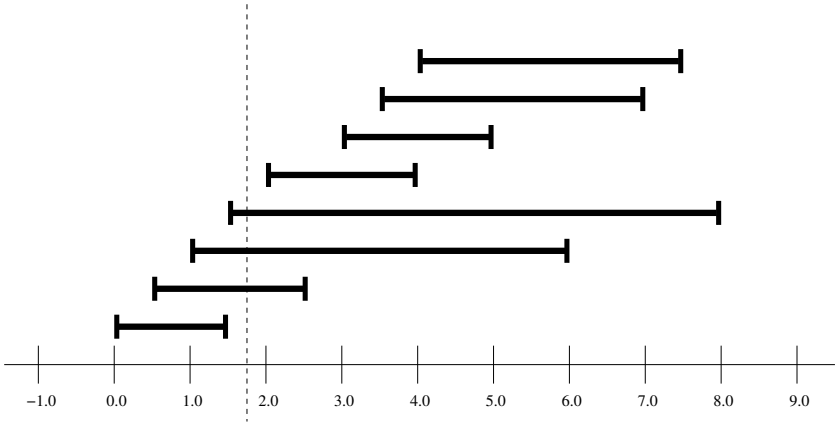

Figura 2: Intervalos generados ordenados según el valor de inicio del intervalo.

Divide y vencerás: La estrategia de *Divide y Vencerás* puede implementarse siguiendo los pasos descritos a continuación.

1. Ordenar el vector de intervalos utilizando la función `mergesortIndInters()`, ver Figura 2.

2. Implementar la función `maxSolDyV()` como una función recursiva donde:
   a) el vector recibido como parámetro está ordenado;
   b) el caso base se da cuando se quiere calcular el solape de un solo intervalo (este solape será 0);
   c) se realizan dos llamadas recursivas: una con la primera mitad de intervalos y otra con la segunda mitad;
   d) se llama a una función auxiliar que calcule el máximo solape entre los intervalos que están en la primera mitad con los intervalos que están en la segunda mitad;
   e) se calcula el máximo de lo que han devuelto estas tres llamadas.

3. Para que la complejidad del algoritmo sea $\mathcal{O}(n \cdot log(n))$, la función auxiliar del punto d) debe calcular los solapes de **un solo intervalo** de la primera mitad con todos los de la segunda mitad. En concreto, el intervalo de la primera mitad que hay que considerar es aquel cuyo valor de `fin` es más alto. Es decir, primero hay que recorrer los intervalos de la primera mitad y guardar el que tiene valor de `fin`

más alto. Luego, se debe calcular el solape de ese intervalo con todos los intervalos de la segunda mitad. De esta manera se calcula el máximo solape entre todos los intervalos de la primera mitad y todos los de la segunda mitad.

6. Código desarrollado en las cuatro primeras prácticas

Como resultado de las cuatro primeras prácticas, cada alumno dispondrá en su cuenta de un directorio (carpeta) denominado **programacion2** dentro del cual se encontrarán los directorios (carpetas) y ficheros que se detallan a continuación.

1. Carpeta **programacion2/funciones** con los siguientes ficheros:
   - Ficheros de interfaz y de implementación, **pilaEnt.hpp** y **pilaEnt.cpp**
2. Carpeta **programacion2/practica0**, con los siguientes ficheros fuentes:
   - Fichero **tiempoReaccion.cpp**.
   - Fichero **generarTabla01.cpp**.
   - Fichero **generarTabla02.cpp**.
   - Fichero **medirCoste.cpp**.
3. Carpeta **programacion2/practica1** con los siguientes ficheros:
   - Ficheros de interfaz y de implementación, **calculos.hpp** y **calculos.cpp**.
   - Ficheros con los programas de prueba (**pruebas01.cpp**, etc.) que se hayan puesto a punto para realizar pruebas de los desarrollos anteriores.
   - Fichero **Make_pruebas01** para compilar los programas de prueba.
4. Carpeta **programacion2/practica2** con los siguientes ficheros:
   - Ficheros de interfaz y de implementación, **funcionesPilaEnt.hpp** y **funcionesPilaEnt.cpp**.
   - Ficheros con los programas de prueba (**pruebas02.cpp**, etc.) que se hayan puesto a punto para realizar pruebas de los desarrollos anteriores.
   - Fichero **Make_pruebas02** para compilar los programas de prueba.
5. Carpeta **programacion2/practica3** con los siguientes ficheros:
   - Los ficheros `maxsolape.hpp`, `maxsolape.cpp` y `calmsolape.cpp`.
   - El fichero `Makefile` que los compila y genera el ejecutable `calmsolape`.

La duración de esta práctica es de dos sesiones. Por tanto, estos ficheros deberán estar listos antes de la sexta sesión de prácticas.