

# PREDA - UNED

Prácticas de curso – 2019/20

# Histórico

Curso	Nº práctica	Esquema	Tema
<b>2011/12</b>	1	DyV	Skyline
	2	RyP	Viajante de comercio
<b>2012/13</b>	1	DyV	Organización calendario liga deportiva
	2	VA	Problema del caballo
<b>2013/14</b>	1	Voraz	Mensajería urgente
	2	PD	Devolución de cambio en monedas
<b>2014/15</b>	1	PD	Multiplicación asociativa de matrices
	2	VA	Coloreado de grafos
<b>2015/16</b>	1	Voraz	Minimización del tiempo en el sistema
	2	PD	Coeficientes binomiales
<b>2016/17</b>	1	DyV	Multiplicación de grandes números
	2	VA	N reinas
<b>2017/18</b>	1	Voraz	Robot en un circuito
	2	PD	Mochila no fraccionable
<b>2018/19</b>	1	VA	Subconjuntos de suma dada
	2	PD	Distancia de edición

# 2019-20

- Práctica 1:
  - Cálculo del elemento mayoritario de un vector (DyV)
- **Práctica 2:**
  - **Reparto equitativo de activos (VA)**

# Normas

- Los estudiantes que tengan aprobadas ambas prácticas en el **curso anterior** (sólo el anterior) no es necesario que vuelvan a realizarlas en este curso.
- Para que el examen sea calificado el alumno deberá:
  - haber asistido a las **sesiones presenciales** de prácticas
  - haber **entregado y aprobado las prácticas** obligatorias
    - Si se entregan y aprueban las prácticas en **septiembre**, hay que presentarse necesariamente al examen de septiembre

# Entrega

- Fechas

- Convocatoria de Febrero:
  - **Domingo 19 de enero de 2020**
- Convocatoria de Septiembre:
  - Domingo 23 de agosto de 2020

1 semana para corregir  
(27/ene: Exámenes)

- **Doble entrega necesaria**

- Por email: [fenros@us.es](mailto:fenros@us.es)
- En la plataforma ALF
  - A través del enlace habilitado en el apartado *"Entrega de trabajos"*

# Entrega

- **¿Todos inscritos al grupo de prácticas del centro asociado de Sevilla?**
  - Plataforma ALF
  - Entrega de trabajos -> Apuntarse a grupo de prácticas

**Lista de alumnos del tutor de Programación y Estructuras de Datos Avanzadas  
Fernando Enríquez de Salamanca en el centro Sevilla**

---

Introduzca su DNI/Pasaporte para apuntarse a la lista de alumnos de este tutor

- **Comprobar que tenéis activadas las dos entregas**

# Evaluación de las prácticas

- Para que se evalúe la práctica es imprescindible que el programa completo compile y funcione
- Su dificultad no será alta, por lo que sólo se calificarán con sobresaliente aquellas prácticas con una excelente documentación y calidad del código, que además cumplan los requisitos imprescindibles de correcto funcionamiento y aplicación de la estructura de datos o esquema adecuados
- En la evaluación que realiza el tutor se tendrán en cuenta los siguientes aspectos:

ASPECTO DE LA PRÁCTICA	NOTA SOBRE 10
Utilización óptima de la estructura de datos o el esquema	2.5
Estilo de programación y calidad del código	2.5
Documentación presentada	2
Eficiencia del algoritmo	2
Posibles mejoras introducidas por el alumno a los requisitos básicos de la práctica	1

# Material a entregar (formato ZIP)

- Archivo .pdf con la siguiente información:
  - Datos de la asignatura
    - Nombre y Código
    - Título de la práctica
    - Centro Asociado
  - Datos del alumno
    - Nombre, apellidos, NIF, teléfono y correo electrónico
  - Respuestas a los apartados:
    - Descripción del esquema utilizado y su adecuación al problema (Seguir las directrices del libro)
    - Análisis del coste computacional
    - Estudio de otras alternativas y comparativa
    - Descripción de los casos de prueba realizados y sus resultados
- Código fuente Java (diseño orientado a objetos)
  - Adecuadamente documentado
  - Ejecutable (.jar) y directorio con fuentes (.java)

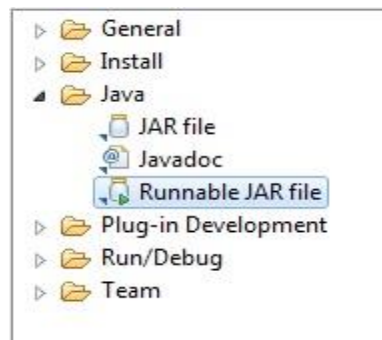


# Crear .jar ejecutable

- BlueJ: Project -> Create Jar File -> Main class



- Eclipse: File -> Export -> Java -> runnable Jar file



# Errores a evitar en la entrega

- Código:
  - No compila
  - No está desarrollado en Java
  - No se corresponde con el libro
  - No es original, está copiado
  - No sigue un diseño OO encapsulado o modular
- Ejecutable:
  - No termina
  - Se queda sin memoria con ejemplos pequeños
  - Aborta sin justificación
  - No lee los ficheros previstos en el formato adecuado
  - No trata los argumentos
  - No se ajusta a las especificaciones
- Documentación:
  - No se presenta como ha indicado el tutor
  - Está incompleta
- Soporte:
  - No se puede leer
  - Contiene virus (Suspenso)

# ¿Dudas?

- Foros de debate de la asignatura en ALF
  - Foro de la práctica
  - Grupo de tutoría de Sevilla
- Correo electrónico al tutor
  - Fernando Enríquez de Salamanca Ros: [fenros@us.es](mailto:fenros@us.es)
- Sesiones de monitorización (obligatorias)
  - Se resolverán colectivamente las dudas más comunes e individualmente las dudas particulares de cada alumno

# Práctica 2

Reparto equitativo de activos

# Enunciado

- Dos socios que forman una sociedad comercial desean disolverla repartiendo a medias sus  $n$  activos que tienen un valor entero positivo. Desean conocer todas las posibles formas que tienen de dividir el conjunto de activos en dos subconjuntos disjuntos de igual valor entero.
- Se pide diseñar un algoritmo, siguiendo el esquema de **vuelta atrás**, que resuelva este problema
- Ejemplo:

Activo	1	2	3	4	5	6	7	8	9	10
Valor	10	9	5	3	3	2	2	2	2	2

Entonces un posible reparto válido sería:

Socio1: 10, 2, 2, 2, 2, 2

Socio2: 9, 5, 3, 3

# Esquema Vuelta Atrás (C6) – Planteamiento

- Aplicación
  - problemas en los que sólo podemos recurrir a una búsqueda exhaustiva, recorriendo el espacio de todas las posibles soluciones hasta encontrar una o hasta que hayamos explorado todas las opciones
- Debido al coste
  - aplicar el conocimiento disponible sobre el problema para abandonar un camino no válido lo antes posible
  - pensar antes si es posible usar un algoritmo voraz
- Recorrido del grafo representa el espacio de búsqueda
  - si el grafo es infinito (o muy grande) se trabaja con un grafo implícito, porque no tiene sentido que el programa lo construya para luego aplicar las técnicas de búsqueda
  - Recorrido en profundidad podando las ramas no válidas
  - Vamos generando una solución parcial (secuencia k-prometedora) y si encontramos un “nodo de fallo” retrocedemos y exploramos otras ramas

# Elementos

- `IniciarExploraciónNivel()`:  
recoge todas las opciones posibles en que se puede extender la solución k-prometedora
- `OpcionesPendientes()`:  
comprueba que quedan opciones por explorar en el nivel
- `SoluciónCompleta()`:  
comprueba que se haya completado una solución al problema
- `ProcesarSolución()`:  
representa las operaciones que se quieran realizar con la solución, como imprimirla o devolverla al punto de llamada
- `Completable()`:  
comprueba que la solución k-prometedora se puede extender con la opción elegida cumpliendo las restricciones del problema hasta llegar a completar una solución

# Esquema

```
fun VueltaAtras (v: Secuencia, k: entero)
    { v es una secuencia k-prometedora }
    IniciarExploraciónNivel(k)
    mientras OpcionesPendientes(k) hacer
        extender v con siguiente opción
        si SoluciónCompleta(v) entonces
            ProcesarSolución(v)
        sino
            si Completable (v) entonces
                VueltaAtras(v, k+1)
            fsi
        fsi
    fmientras
ffun
```



# Esquema

```
fun VueltaAtras (v: Secuencia, k: entero)
    { v es una secuencia k-prometedora }
    IniciarExploraciónNivel(k)
    mientras OpcionesPendientes(k) hacer
        extender v con siguiente opción
        si SoluciónCompleta(v) entonces
            ProcesarSolución(v)
        sino
            si Completable (v) entonces
                VueltaAtras(v, k+1)
            fsi
        fsi
    fmientras
ffun
```

# Esquema

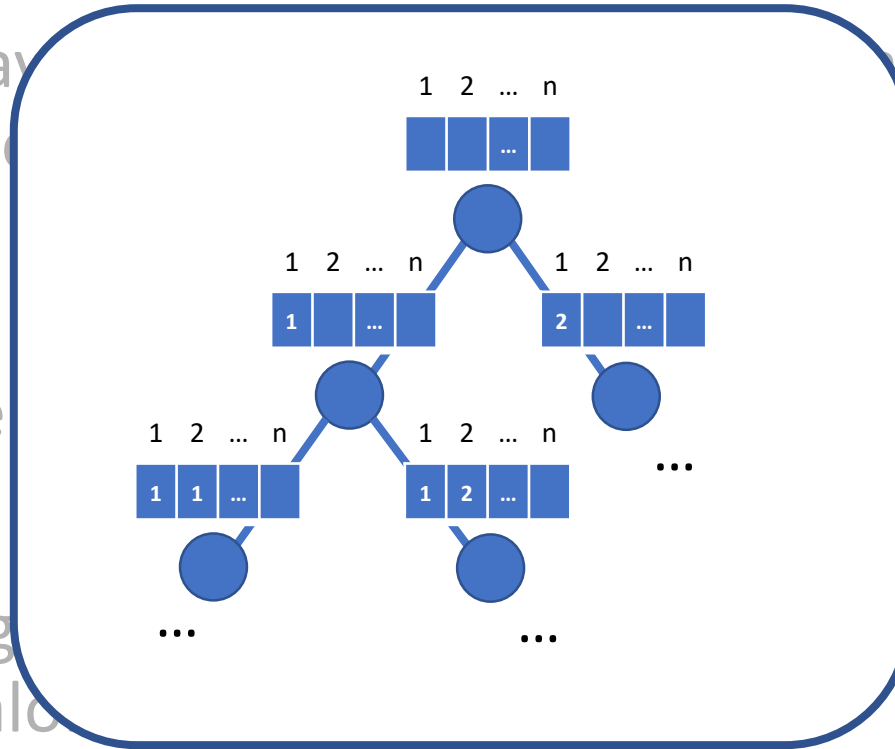
```
fun VueltaAtras (v: Secuencia, k: entero)
  { v es una secuencia k-prometedora }
  IniciarExploraciónNivel(k)
  mientras OpcionesPendientes(k) hacer
    extender v con siguiente opción
    si SoluciónCompleta(v) entonces
      ProcesarSolución(v)
    sino
      si Completable (v) entonces
        VueltaAtras(v, k+1)
      fsi
    fsi
  fmientras
ffun
```

# Reparto equitativo de activos

- Objetivo: ayudar a dos socios que forman una sociedad comercial a disolverla, buscando todas las formas de repartir en dos subconjuntos disjuntos de igual valor sus  $n$  activos con valor entero
- Espacio de búsqueda: árbol de grado 2 y altura  $n+1$
- Se puede generalizar a un reparto en  $n$  subconjuntos de igual valor
  - <https://www.geeksforgeeks.org/partition-set-k-subsets-equal-sum/>

# Reparto equitativo de activos

- Objetivo: ay... una sociedad co... o todas las formas de... disjuntos de igual valor
- Espacio de... altura  $n+1$
- Se puede g... subconjuntos de igual valor



- <https://www.geeksforgeeks.org/partition-set-k-subsets-equal-sum/>

**tipo** Vector = matriz[0..N] de entero

**fun** DividirSociedad(x: Vector, suma1, suma2, sumaTotal, k: entero, v: Vector)

{ v es un vector k-prometedor }

**si** k = N **entonces**

**si** suma1 = suma2 **entonces**

        Procesar(v)

**fsi**

**sino**

    v[k+1] ← 1

**si** Completable (x, suma1, sumaTotal, k+1) **entonces**

        suma1 ← suma1 + x[k+1]

        DividirSociedad(x, suma1, suma2, sumaTotal, k+1, v)

**fsi**

    v[k+1] ← 2

**si** Completable (x, suma2, sumaTotal, k+1) **entonces**

        suma2 ← suma2 + x[k+1]

        DividirSociedad(x, suma1, suma2, sumaTotal, k+1, v)

**fsi**

**fsi**

**ffun**

- **x**: vector de activos
- **suma1, suma2, sumaTotal**: suma de activos asignados a cada socio y el total
- **v**: vector de asignaciones

Falta (con suma2 también):  
suma1 ← suma1 - x[k+1]

**fun** Completable(x:Vector, sumaParcial, sumaTotal, k: entero): booleano

**si** sumaParcial + x[k] ≤ sumaTotal **div** 2 **entonces**

**dev** cierto

**sino**

**dev** falso

**fsi**

**ffun**

Ojo a las erratas  
del libro

**tipo** Vector = matriz[0..N] de entero

**fun** DividirSociedad(x: Vector, suma1, suma2, sumaTotal, k: entero, v: Vector)

{ v es un vector k-prometedor }

**si** k = N **entonces**

**si** suma1 = suma2 **entonces**

        Procesar(v)

**fsi**

**sino**

    v[k+1] ← 1

**si** Completable (x, suma1, sumaTotal, k+1) **entonces**

        suma1 ← suma1 + x[k+1]

        DividirSociedad(x, suma1, suma2, sumaTotal, k+1, v)

**fsi**

    v[k+1] ← 2

**si** Completable (x, suma2, sumaTotal, k+1) **entonces**

        suma2 ← suma2 + x[k+1]

        DividirSociedad(x, suma1, suma2, sumaTotal, k+1, v)

**fsi**

**fsi**

**ffun**

- **x**: vector de activos
- **suma1, suma2, sumaTotal**: suma de activos asignados a cada socio y el total
- **v**: vector de asignaciones

Falta (con suma2 también):  
suma1 ← suma1 - x[k+1]

Cota superior de coste:  
forma del árbol =  $O(2^n)$

**fun** Completable(x:Vector, sumaParcial, sumaTotal, k: entero): booleano

**si** sumaParcial + x[k] ≤ sumaTotal **div** 2 **entonces**

**dev** cierto

**sino**

**dev** falso

**fsi**

**ffun**

Ojo a las erratas  
del libro

**tipo** Vector = matriz[0..N] de entero

**fun** DividirSociedad(x: Vector, suma1, suma2, sumaTotal, k: entero, v: Vector)

{ v es un vector k-prometedor }

**si** k = N **entonces**

**si** suma1 = suma2 **entonces**

        Procesar(v)

**fsi**

**sino**

    v[k+1]  $\leftarrow$  1

**si** Completable (x, suma1,

        suma1  $\leftarrow$  suma1 + x[k]

        DividirSociedad(x, sur

**fsi**

    v[k+1]  $\leftarrow$  2

**si** Completable (x, suma2,

        suma2  $\leftarrow$  suma2 + x[k]

        DividirSociedad(x, sur

**fsi**

**fsi**

**ffun**

**fun** ResolverSeparacionSocios (x:Vector)

**var**

        i, suma1, suma2, sumaTotal: entero

        v: Vector

**fvar**

        sumaTotal  $\leftarrow$  0

        suma1  $\leftarrow$  0

        suma2  $\leftarrow$  0

**para** i  $\leftarrow$  1 **hasta** N **hacer**

        sumaTotal  $\leftarrow$  sumaTotal + x[i]

**fpara**

**si** sumaTotal **mod** 2 = 0 **entonces**

            DividirSociedad(x,suma1,suma2, sumaTotal,0,v)

**fsi**

**ffun**

**fun** Completable(x:Vector, sumaParcial, sumaTotal, k: entero): booleano

**si** sumaParcial + x[k]  $\leq$  sumaTotal **div** 2 **entonces**

**dev** cierto

**sino**

**dev** falso

**fsi**

**ffun**

# Coste

- El coste del caso peor es del orden del tamaño del espacio de búsqueda
- Las funciones de poda que utilicemos reducen el coste, aunque muchas veces no es posible saber cuanto (depende de los datos), así que se da una cota superior



# Argumentos y parámetros

- Sintaxis:
  - `java reparto [-t][-h] [fichero_entrada] [fichero_salida]`
  - `java -jar reparto.jar [-t][-h] [fichero_entrada] [fichero_salida]`
- Argumentos:
  - `-t`: traza cada paso de manera que se describa la aplicación del algoritmo utilizado
  - `-h`: muestra una ayuda y la sintaxis del comando

Ejemplo:

```
$ java reparto -h <ENTER>
SINTAXIS:
reparto [-t][-h] [fichero_entrada] [fichero_salida]
-t                Traza las llamadas recursivas
-h                Muestra esta ayuda
fichero_entrada   Nombre del fichero de entrada
fichero_salida    Nombre del fichero de salida
```

- `fichero_entrada`: nombre del fichero del que se leen los datos de entrada
  - Contiene el número de activos, seguido del valor de cada activo
  - Si la entrada no es correcta, el programa debe indicarlo
- `fichero_salida`: es el nombre del fichero que se creará para almacenar la salida
  - Si el fichero ya existe, el comando dará un error
  - Si falta este argumento, el programa muestra el resultado por pantalla

# Argumentos y parámetros

- Sintaxis:

```
public class Reparto{  
    ...  
    public static void main(String[] args){  
        switch (args.length){  
            case 0: //no hay argumentos  
                break;  
            case 1: //args[0] puede ser -t, -h, fich_ent o fich_sal  
                break;  
            case 2: //args[0] y args[1]  
                break;  
            case 3: //args[0], args[1], args[2]  
                break;  
            case 4: //args[0], args[1], args[2], args[3]  
                break;  
            default: //demasiados argumentos  
                break;  
        }  
    }  
    ...  
}
```

ado

# Entrada / Salida


- Entrada

- El fichero de datos de entrada consta de 2 líneas con:
  - El valor del parámetro n (número de activos)
  - Los valores de los activos (sin ordenar) separados por espacios
- Ejemplo:  
10  
10 9 5 3 3 2 2 2 2 2

- Salida

- La salida es:
  - Una línea con el número de soluciones encontradas
  - Tres líneas por cada solución:
    - Identificador de la solución
    - Los valores de los activos asignados al socio1
    - Los valores de los activos asignados al socio2

- Ejemplo (correspondiente al ejemplo de entrada):



```
6
1
10 5 3 2
9 3 2 2 2 2
2
10 3 3 2 2
9 5 2 2 2
3
10 2 2 2 2 2
9 5 3 3
4
9 3 2 2 2 2
10 5 3 2
5
9 5 2 2 2
10 3 3 2 2
6
9 5 3 3
10 2 2 2 2 2
```

# Código auxiliar

Fragmentos de código Java que pueden ser útiles

# Lectura de ficheros en java (I)

- Lectura de un fichero de texto línea a línea:

```
BufferedReader bf = new BufferedReader(new FileReader(nomFich));
```

```
String linea = bf.readLine();
```

```
// Si no quedan datos en el fichero, linea será null
```

```
// Para dividir una línea:
```

```
String[] datos = linea.split(" ")
```

```
// Para convertir a entero un String:
```

```
volumen = new Integer(datos[0])
```

```
// Hay que cerrar el fichero una vez leído
```

```
bf.close();
```

# Lectura de ficheros en java (II)

- Scanner

```
Scanner sc = new Scanner(new File(nomFich));  
while(sc.hasNextLine()){  
    String linea = sc.nextLine();  
    //procesar elemento  
}  
sc.close();
```

# Escritura de ficheros en java

- `PrintStream`

```
PrintStream ps = new PrintStream(new File(nomFich));
```

```
...
```

```
ps.println(elem); //ps.print(elem + "\n");
```

```
...
```

```
ps.close();
```