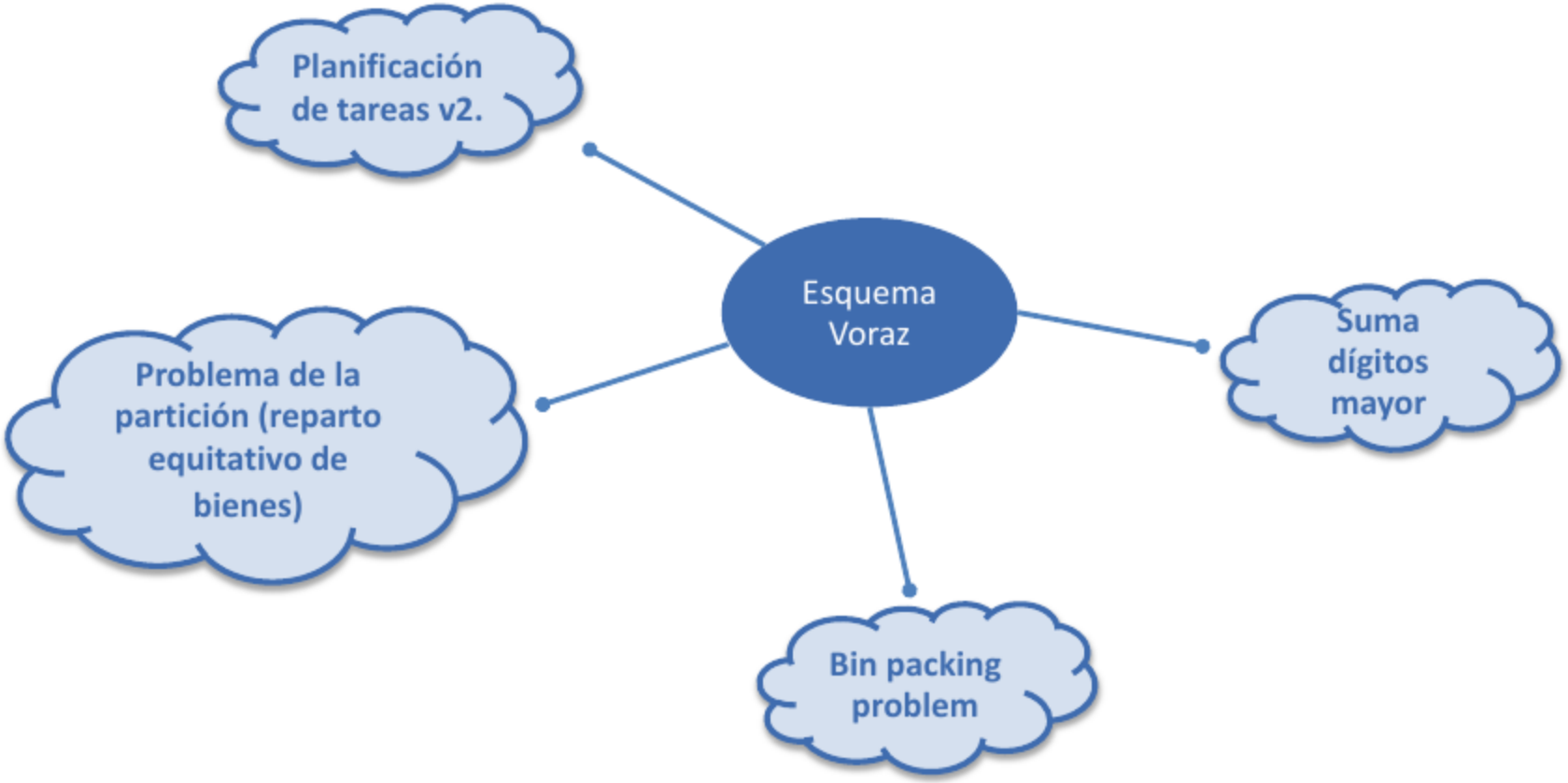
Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Tema 12. Ejemplos algoritmos voraces

Algorítmica y Complejidad

Algunos problemas planteados



Algunos problemas planteados

Enunciado

- Se tiene un conjunto de tareas (por ej., procesos en un SO) que deben usar un recurso (por ej., el procesador) que sólo puede ser usado por una actividad en cada instante.
- Todas las tareas requieren 1 unidad de tiempo para ejecutarse
- Cada tarea *i* tiene asociado:
  - un plazo máximo de ejecución d_i y un beneficio b_i, que se obtiene si la tarea consigue llegar a ejecutarse. Una tarea sólo puede ejecutarse si se hace en un plazo inferior a d_i.
- Objetivo: Obtener una planificación de tareas (qué tareas y en qué orden) a ejecutar de forma que se maximice el beneficio obtenido.

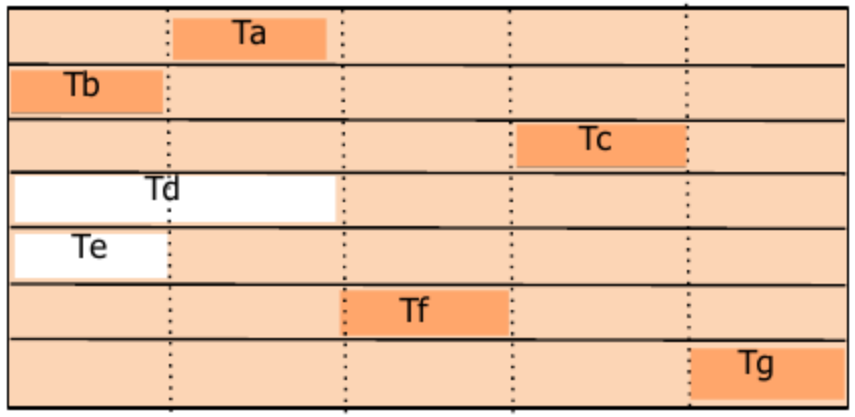
Esquema selección óptima
$\Theta(2^N)$

Esquema voraz:
$\Theta(N)$, $\Theta(N \lg N)$ o $\Theta(N^2)$

Implementación

- Opciones para la propiedad voraz:
  - ✗ Elegir las tareas por orden de terminación
    (b/e, a/d, c, f, g) o (b/e, a/d, f, c, g)
  - ✗ Elegir las tareas por beneficio (a, f, c, g).
  - ✓ Elegir las tareas por orden de terminación y beneficio.

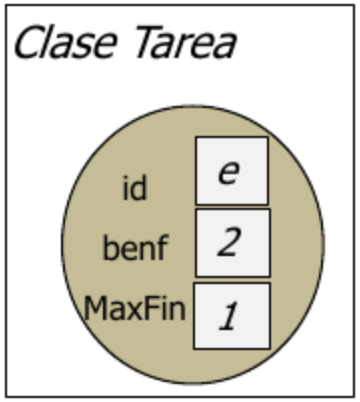
b_i	d_i
80	2
10	1
15	4
27	2
2	1
44	4
1	5



Implementación

- Clase Tarea contiene la información relativa a una Tarea concreta:

```
public class Tarea{
  private char id;
  private int beneficio;
  private int maxFin;

  Tarea(char id, int beneficio, int maxFin){
    this.id  = id;
    this.beneficio  = beneficio;
    this.maxFin = maxFin;
  }
  /* getters y setters */
}
```



ArrayList<Tarea> **candidatos**;

ArrayList<Tarea> **solucion**;

- Conjunto de candidatos: tareas disponibles.
- Conjunto de seleccionados: las tareas a las que se puede dar servicio.

Implementación

- **Función Objetivo**: maximizar el beneficio total.
- **Enfoque voraz**: cuanto antes termine una tarea más posibilidades habrá de planificar más tareas y obtener más beneficio.
- **Función Selección**: consiste en la selección de aquella tarea pendiente cuyo plazo de terminación sea menor y, si hay varias tareas en este grupo, seleccionar la que aporte más beneficio.
- **Función Factibilidad**: sólo serán válidas las tareas cuyo plazo máximo de ejecución no haya expirado.
- **Función Solución**: cualquier planificación es válida, siendo la óptima aquella que maximice el beneficio total.

Implementación

```
private Tarea seleccionarCandidatoListaDesordenada(ArrayList<Tarea> candidatos){
    Tarea mejor = null;
    for (Tarea tarea : candidatos) {
        if (mejor == null){
            mejor = tarea;
        }
        else if (mejor.getMaxFin() > tarea.getMaxFin()) {
            mejor = tarea;
        }
        else if ((mejor.getMaxFin() == tarea.getMaxFin()) &&
                 ((mejor.getBeneficio() < tarea.getBeneficio()))){
                // la tarea actual termina en el mismo momento que la mejor,
                // pero aporta más beneficio
                mejor = tarea;
            }
        }
        return mejor;
    }
```

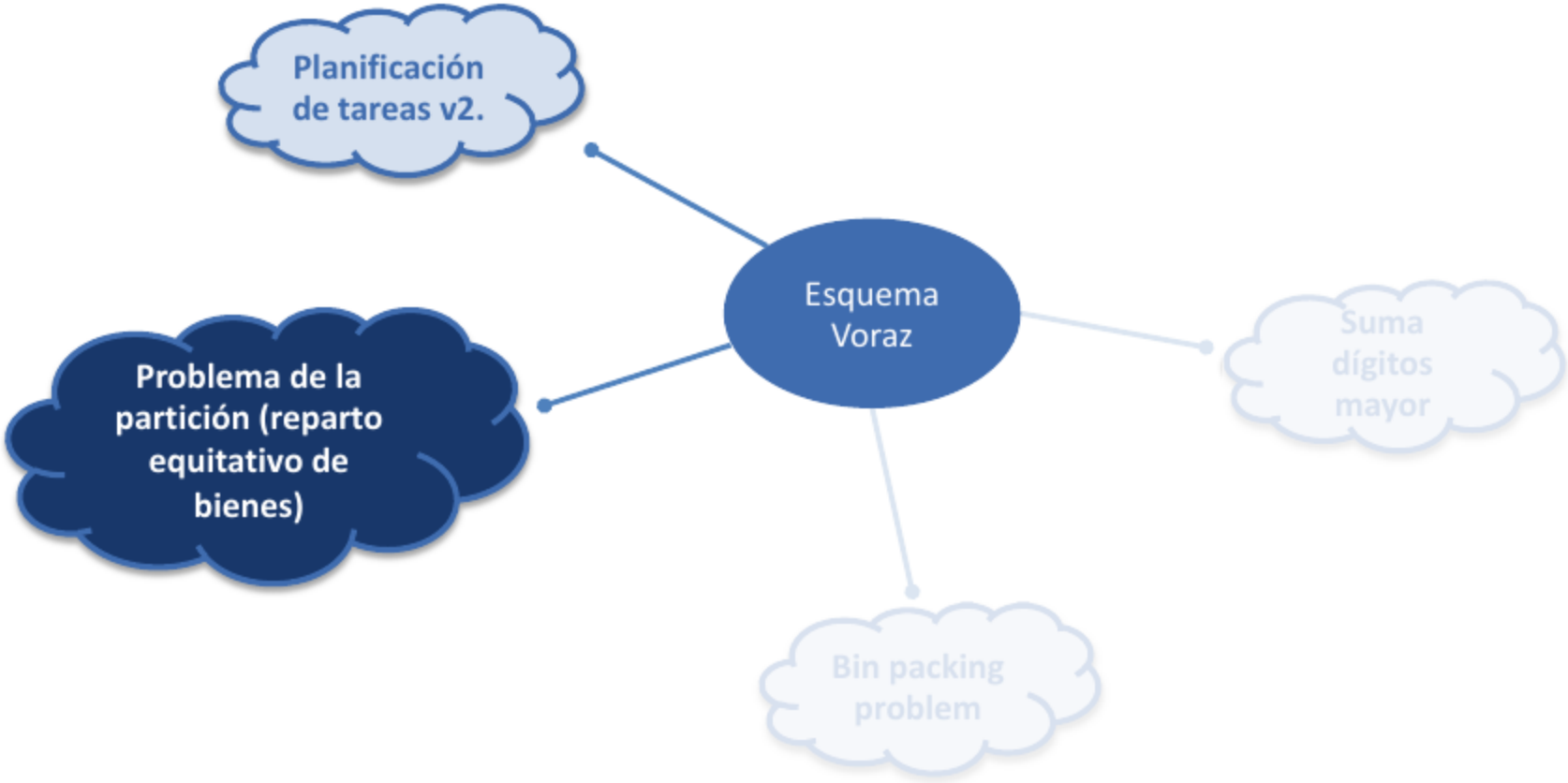
Implementación

```
private boolean esFactibleCandidato(Tarea candidato,
                                    ArrayList<Tarea> solucion) {
    // Un candidato es factible si su plazo máximo de ejecución no ha expirado.
    // El tiempo actual viene determinado por el número de tareas planificadas
    // hasta el momento (número de elementos en solucion)

    boolean esFactible = true;

    if (solucion.size()>0)
        if (candidato.getMaxFin() <= solucion.size()) esFactible= false;
    return esFactible;
}
```

Implementación

```
public ArrayList<Tarea> algoritmoVoraz(ArrayList<Tarea> candidatos) {
    // Inicializar solución
    ArrayList<Tarea> solucion = new ArrayList<Tarea>();
    Tarea candidato = null;
    while (! candidatos.isEmpty()) { // mientras queden candidatos
        // Seleccionar candidato
        candidato = seleccionarCandidatoListaDesordenada(candidatos);
        candidatos.remove(candidato); // Eliminar candidato
        if (esFactibleCandidato(candidato, solucion))
            // Añadir candidato a la solución
            solucion.add(candidato);
    }

    if (solucion.size() > 0) // se ha conseguido planificar al menos 1 tarea
        return solucion;
    else return null;
  }
```

Algunos problemas planteados



Enunciado

- Se tiene un conjunto de bienes valorados y dos herederos entre los que repartir los bienes.

- Objetivo: Repartir los bienes entre los dos herederos de manera equitativa (la suma de valores de los bienes de cada heredero debe ser igual).

Esquema backtracking
$\Theta(2^N)$

Esquema programación dinámica:
$\Theta(N\ k)$ pseudopolinomial

Esquema voraz:
$\Theta(N)$, $\Theta(N\ \lg N)$ o $\Theta(N^2)$

No garantiza encontrar la solución óptima!!!

Implementación

- Función Objetivo: repartir los bienes de forma equitativa.
- Enfoque voraz: ir asignando los bienes de mayor valor al heredero que menos valor acumulado tenga, de forma que se intenta ir equilibrando el reparto.
- Función Selección: seleccionar el bien de mayor valor.
- Función Factibilidad: el bien seleccionado siempre es aceptable (se asigna al heredero que menos valor acumulado tenga en cada momento).
- Función Solución: aquella que distribuya los bienes de forma equitativa.

Implementación

```
private int seleccionarCandidato(ArrayList<Integer> candidatos){
    // selecciona el candidato que mayor valor tenga

    int c = candidatos.get(0);
    for (int i=1; i<candidatos.size();i++)
        if (c < candidatos.get(i)) c=candidatos.get(i);
    return c;
}
```

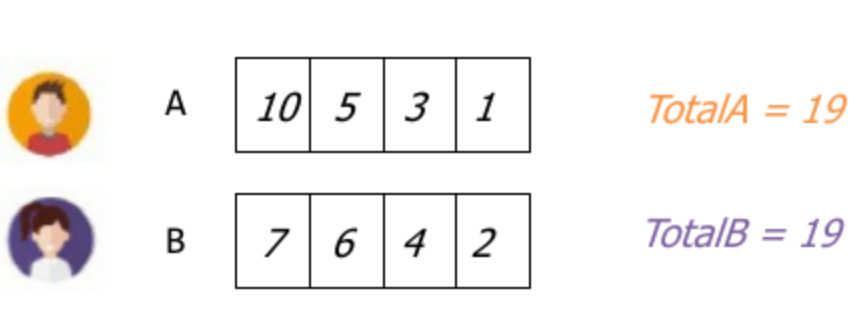
Implementación

```
public void algoritmoVoraz(ArrayList<Integer> bienes,
                           ArrayList<Integer> A,
                           ArrayList<Integer> B){
int totalA=0, totalB=0, c;

    while (!bienes.isEmpty()){
        c = seleccionarCandidato(bienes);
        bienes.remove(new Integer(c));
        if (totalA > totalB) {
            B.add(c);    totalB = totalB + c;
        }
        else{
            A.add(c);    totalA = totalA + c;
        }
    }
}
```

Implementación

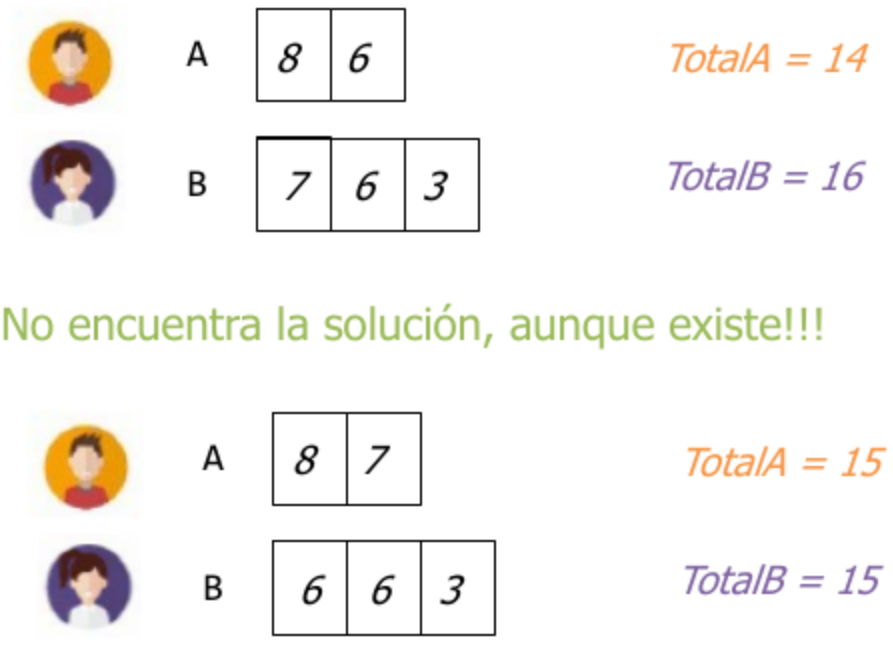
```
ArrayList<Integer> bienes;
```

2	4	1	7	10	6	3	5

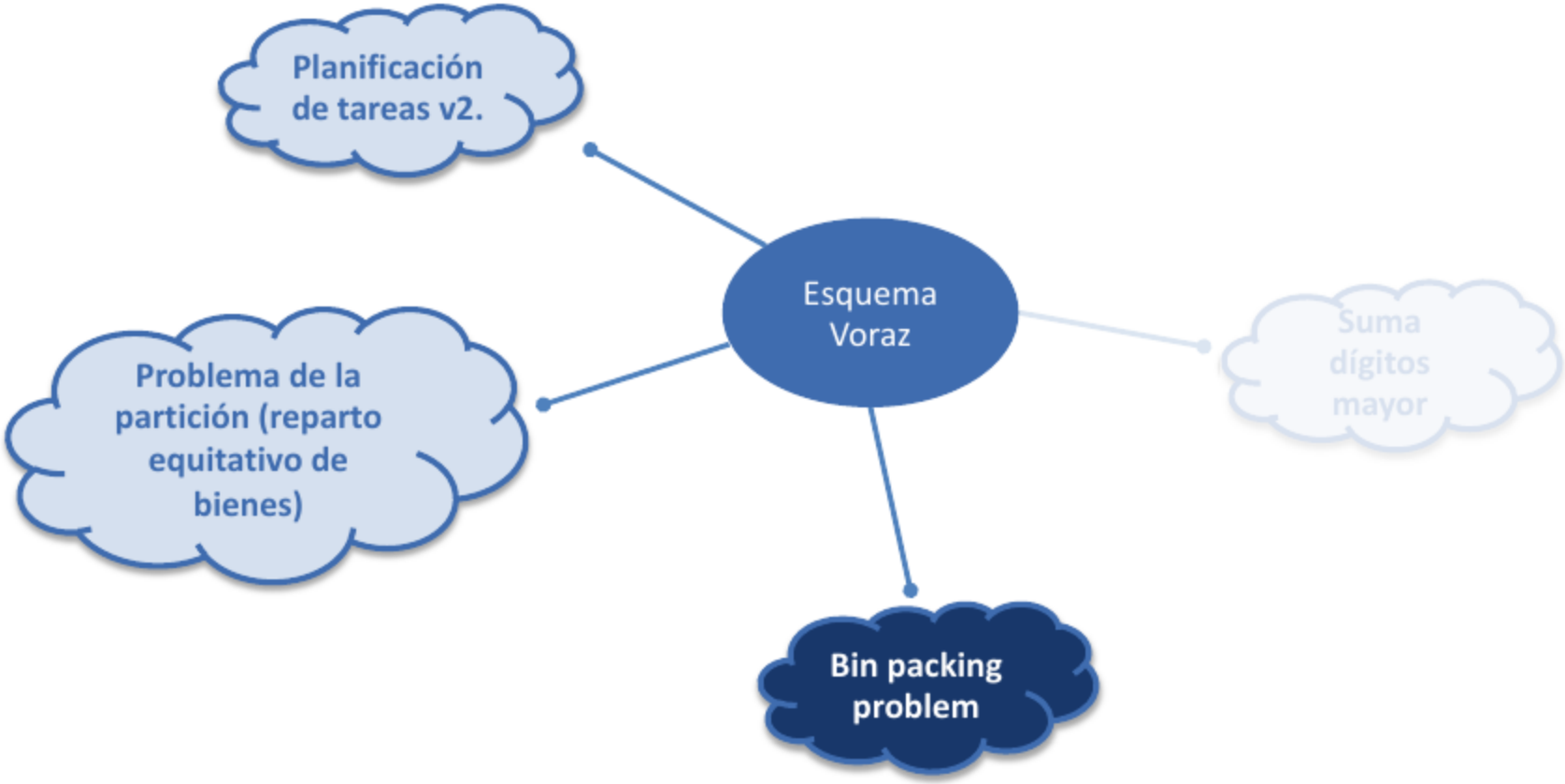



```
ArrayList<Integer> bienes;
```

7	6	6	8	3



Algunos problemas planteados



Enunciado

- Se tiene un conjunto de items de diferentes pesos y un conjunto de contenedores, cada uno con capacidad C.

- Objetivo: Minimizar el número de contenedores para empaquetar todos los items, asegurando que ningún contenedor sobrepasa su peso máximo.

Esquema selección óptima
$\Theta(N^N)$

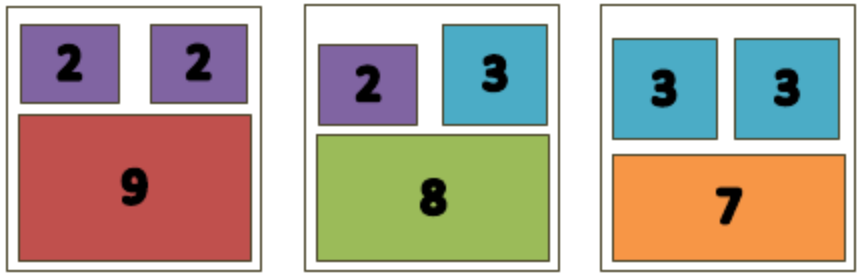
Esquema voraz:
$\Theta(N)$, $\Theta(N \lg N)$ o $\Theta(N^2)$

No garantiza encontrar la solución óptima!!!

C = 13

items

2	9	3	2	8	7	3	2	3



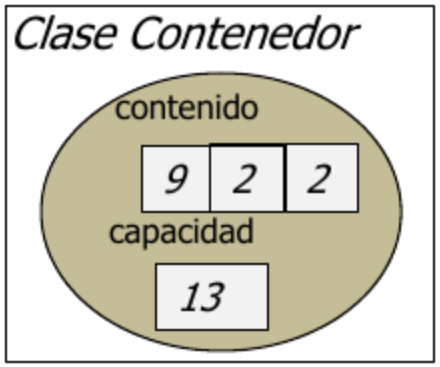
Implementación

- Función Objetivo: minimizar el número de contenedores para empaquetar todos los items.
- Enfoque voraz: ir empaquetando los items de mayor peso en los contenedores (si no cabe en alguno de los usados hasta el momento se asigna uno nuevo), de forma que los items finales (los de menor peso) rellenen los huecos restantes.
- Función Selección: seleccionar el item de mayor peso.
- Función Factibilidad: el item seleccionado siempre es aceptable (se asigna a uno de los contenedores en los que quepa o se selecciona un nuevo contenedor).
- Función Solución: la distribución de todos los items en contenedores. Será óptima si minimiza el número de contenedores.

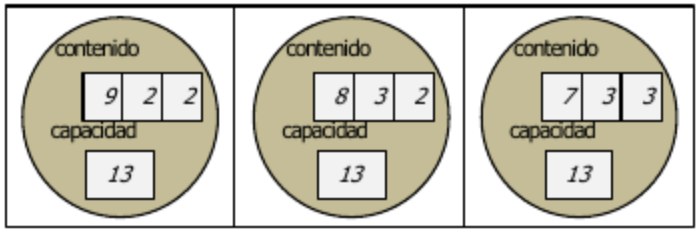
Implementación

- Clase Contenedor contiene la información relativa al contenido de un contenedor:

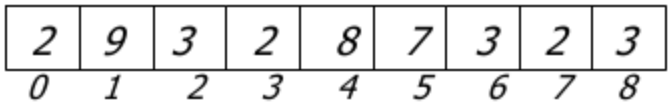
```
public class Contenedor{
   private ArrayList<Integer> contenido;
   private int capacidad;

   Contenedor(int capacidad){
     this.capacidad  = capacidad;
     this.contenido = new ArrayList<Integer>();
   }

   public int getPesoRestante(){
     int p = 0;
     for (int i=0; i<contenido.size(); i++)
         p = p + contenido.get(i);
     return this.capacidad-p;
   }

   public void aniadeItem(int item){
     this.contenido.add(item);
   }
  /* getters y setters */
}
```



ArrayList<Contenedor> **solucion**;



ArrayList<Integer> **candidatos**;



Implementación

```
private int seleccionarCandidato(ArrayList<Integer> candidatos){
   // selecciona el candidato que mayor valor tenga

   int c = candidatos.get(0);
   for (int i=1; i<candidatos.size();i++)
       if (c < candidatos.get(i)) c=candidatos.get(i);
   return c;
}
```

Implementación

```
private void aniadirCandidato(int item, ArrayList<Contenedor> contenedores,
                              int Capacidad){
    boolean encontrado = false;
    int c=0;
    Contenedor Cont;
    while ((c < contenedores.size()) && !encontrado){
        if (contenedores.get(c).getPesoRestante() >= item){
            encontrado=true;
            contenedores.get(c).aniadeItem(item);
        }
        c++;
    }
    if (!encontrado){ // añadimos un nuevo contenedor
        Cont = new Contenedor(Capacidad);
        Cont.aniadeItem(item);
        contenedores.add(Cont);
    }
}
```

Implementación

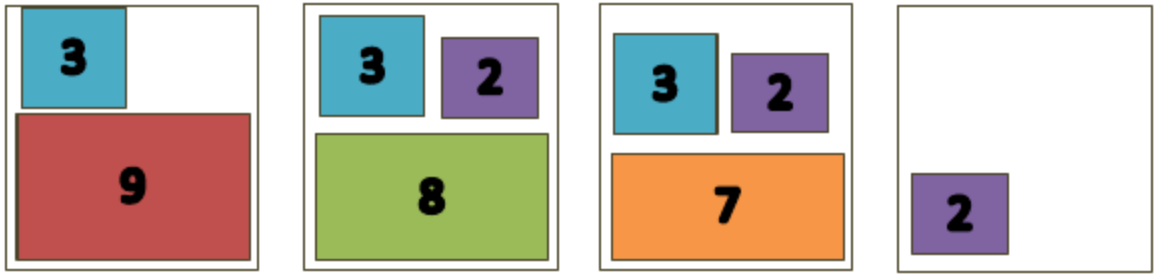
```
public ArrayList<Contenedor> esquemaVoraz(ArrayList<Integer> pesos,
                                          int C){

    ArrayList<Contenedor> contenedores = new ArrayList<Contenedor>();
    int item;

    while (!pesos.isEmpty()){
        item = seleccionarCandidato(pesos);
        pesos.remove(new Integer(item));
        // el candidato siempre es aceptable. Se comprueba si cabe en
        // los contenedores abiertos (y no llenos) hasta el momento
        // y si no es así se selecciona un nuevo contenedor
        aniadirCandidato(item, contenedores, C);
    }
    return contenedores;
}
```

Implementación

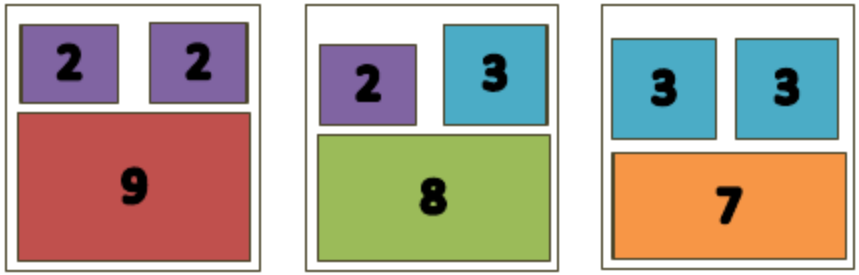
C = 13

items

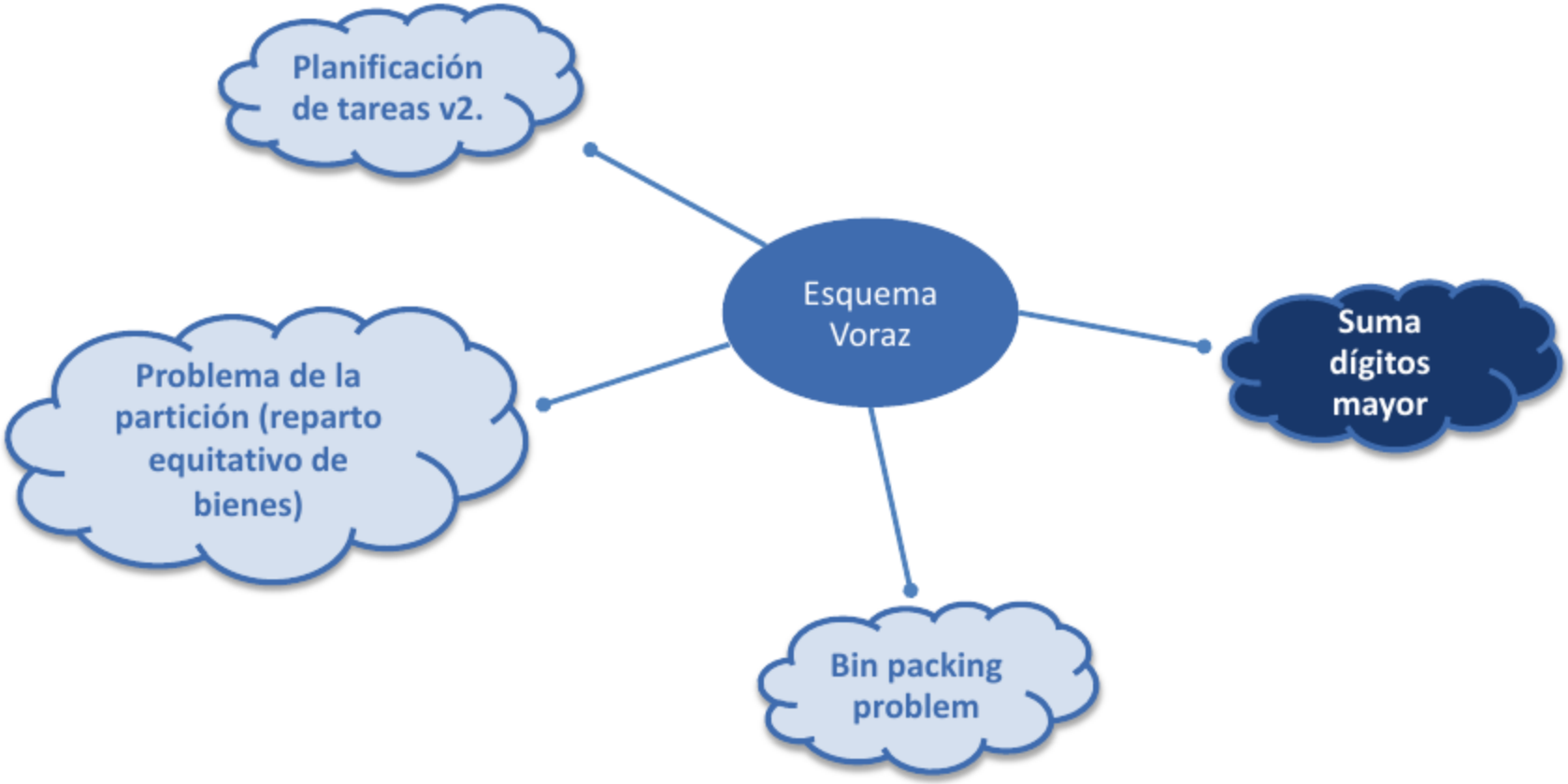
2	9	3	2	8	7	3	2	3



No garantiza encontrar la solución óptima!!!



Algunos problemas planteados



Enunciado

- Dados dos números enteros positivos S y D

Esquema voraz: Θ(N)

- Objetivo: Encontrar el número más grande que contenga D dígitos y cuya suma de dígitos sea S.

D

S \ D	1	2	3	4	5	6	…
0	✖	✖	✖	✖	✖	✖	…
1	1	10	100	1000	10000	100000	…
9	9	90	900	9000	90000	900000	…
15	✖	96	960	9600	96000	960000	…
24	✖	✖	996	9960	99600	996000	…
36	✖	✖	✖	9999	99990	999900	…
…	…	…	…	…	…	…	…

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

*Si el valor de S está fuera del rango [1..9*D] no existe solución*

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

Vector para almacenar la solución, con tantos dígitos como indique D

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

candidato: *dígito candidato (9…0)*
d: *posición de "solucion" en la que incluir el próximo dígito*
c: *nº de dígitos de tipo "candidato" a incluir en la solución*

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

fin: indicará cuándo hemos rellenado todos los dígitos

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

Mientras queden dígitos por rellenar

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

Nº de dígitos de tipo "candidato" que podrían incluirse en la "solucion"

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

Si el candidato es aceptable (al menos se puede incluir uno de los dígitos de tipo "candidato")

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

Rellena "solucion" con tantos dígitos de tipo "candidato" como indique la variable "c", incrementando la posición de "solucion" (d) en cada inserción

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

Resta a S el valor de los dígitos recién insertados

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

Si ya no podemos incluir más dígitos mayores que 0

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

Rellena el resto de posiciones de "solucion" con 0's, e indica que ya se ha encontrado la solución

Implementación

```
public int[] esquemaVoraz(int S, int D){
    if ((S<1) || (S> 9*D)) return null;
    else {
        int[] solucion = new int[D];  // vector para la solucion
        int candidato = 9,   d = 0,   c;
        boolean fin = false;
        while (!fin) {
            c = S / candidato;  // c -> Nº de digitos "candidato" a incluir
            if (c > 0) {
                for (int i = 0; i < c; i++) {solucion[d] = candidato;   d++; }
                S = S - (candidato * c);
                if (S == 0) {
                    for (int i = d; i < D; i++) solucion[i] = 0;
                    fin = true;
                }
            }
            candidato--;
        }
        return solucion;
    }
}
```

Se desecha el candidato recién tratado y se selecciona el siguiente candidato