Universidad Autónoma Metropolitana Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Reporte Final del Proyecto Terminal

Asignación óptima para el uso de laboratorios y talleres de la División de Ciencias Básicas e Ingeniería

Villarruel Barajas Antonio Eduardo
207300364

Trimestre 12 – O

M. en C. Rodrigo Alexander Castro Campos Dr. Francisco Javier Zaragoza Martínez

Tabla de Contenido

Asignación óptima para el uso de laboratorios y talleres de la División de Ciencias Básicas e Ingeniería

Resumen

El presente documento tiene como objetivo describir los avances que se realizaron a lo largo del trimestre 12 – O para la conclusión de la presente propuesta, la cual dado el ingreso de datos de disponibilidad de los laboratorios y talleres propios de la División de CBI, ejecuta un algoritmo que realiza la asignación óptima de dichos espacios bajo las restricciones y demandas que posea cada uno.

Asimismo, el reporte presenta un conjunto de documentos que buscan facilitar el modo de empleo del proyecto desarrollado a través de ejemplos de uso, manual de instalación y manual del usuario entre otros.

Objetivo general.

Diseñar e implementar un sistema que realice una asignación óptima de los laboratorios y talleres de la División de CBI de la UAM Azcapotzalco.

Objetivos específicos.

- Diseñar e implementar una interfaz web que permita ingresar los datos de disponibilidad de los laboratorios y talleres de CBI.
- Diseñar e implementar una interfaz web que permita especificar la demanda de grupos que requieran el uso de laboratorios y talleres.
- Diseñar e implementar un algoritmo que realice una asignación óptima de los espacios físicos indicados, bajo las restricciones y demandas dadas

Introducción.

En las ingenierías de la UAM Azcapotzalco, algunas UEA[1] tienen una componente práctica la cual requiere del uso de espacios físicos[2] especialmente diseñados. La subutilización de dichos espacios limita fuertemente la cantidad de grupos que se pueden ofrecer a los alumnos que planean cursar dichas UEA.

[1] Unidades de Enseñanza-Aprendizaje.

[2] Se entiende por espacios físicos a aquellos laboratorios y talleres propios de la División de CBI.

La asignación de estos espacios físicos está sujeta a un conjunto de restricciones tales como el horario de trabajo de los encargados de dichos espacios o el tiempo de preparación necesario para poder hacer uso de ellos.

Actualmente la asignación de grupos en laboratorios y talleres es un proceso que se realiza de forma manual e involucra reunir a diversos representantes de la División con el fin de llevar a cabo la programación trimestral de dichos espacios.

Este proyecto busca implementar un sistema web para el ingreso de datos de disponibilidad de los laboratorios y talleres propios de la División de CBI, además de diseñar e implementar un algoritmo que realice la asignación óptima de dichos espacios bajo las restricciones y demandas que posea cada uno.

Calendario de Trabajo del trimestre 12-O

A continuación se presenta el calendario de actividades planteado en la propuesta para el trimestre 12-O, el cual contempló 18 horas de trabajo por semana durante todo el periodo trimestral:

Proyecto Terminal II **Trimestre 12-O**	**Semana**										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Diseño e implementación del algoritmo de asignación óptima.											
Evaluación del desempeño del algoritmo.											
Aplicación de pruebas y correcciones del algoritmo de asignación óptima.											
Elaboración del reporte final y documentación entregable.											

Tabla 1. Calendario de Trabajo para el trimestre 12-O.

Reporte de las actividades realizadas

Semanas 1, 2 y 3 [Del 10 al 28 de Septiembre de 2012]

Actividades Realizadas:

- Creación de las primeras clases en Java para el almacenamiento de la información enviada por PHP.

Descripción:

Durante estas semanas, se realizaron las primeras clases para el almacenamiento e interpretación de los datos enviados por los módulos de *gestión de datos de disponibilidad* y *gestión de demanda de los espacios físicos*. La primera clase que se creó fue *Dupla*, su objetivo es el almacenamiento de un intervalo de tiempo de un espacio físico del cual podemos hacer uso; sus atributos son el minuto de *inicio* y el minuto *fin*, algunos de los métodos que se crearon fueron el método *duración* y *menorQue*, el cual implementa la interface *Comparator*.

Otra de las clases que se crearon fue *Horarios*, el cual consta de una lista doblemente enlazada de objetos *Dupla* y un entero que almacena el tiempo de preparación necesario para poder hacer uso de dicho espacio; cabe destacar la creación del método *simplifica_duplas*, que antes de realizar el llenado de la lista doblemente enlazada, se encarga de ordenar y simplificar las duplas, fusionando aquellas que pueden reducirse en una sola.

Es importante mencionar que la lista doblemente enlazada utilizada por este proyecto fue desarrollada desde cero, ya que la lista que posee el API de Java no contiene algunos métodos que se iban a necesitar para el funcionamiento del algoritmo o bien son ineficientes (como lo son los métodos *merge* y *splice*); entre los principales métodos que se implementaron están: *insertarEnCabeza*, *insertarDespuesDe, next, previous, splice, merge* y *merge_front*.

Semanas 4, 5 y 6 [Del 1° al 19 de Octubre de 2012]

Actividades Realizadas:

- Continuación de la construcción de clases en Java para el almacenamiento de la información enviada por PHP.

- Análisis de las estrategias de programación dinámica y fuerza bruta para el diseño del algoritmo.

Descripción:

En este periodo se desarrolló la clase *UEA* para el manejo de los datos de demanda, la cual almacena toda la información de las UEA que se desean asignar, sus atributos son el número de grupos requeridos, número de sesiones por semana, la duración de cada sesión y los salones en los que se puede tomar dicha UEA.

Posteriormente, se realizaron los primeros acercamientos del algoritmo de asignación. Aunque inicialmente se había planteado que el algoritmo se basara en programación dinámica, al calcular la complejidad que podía alcanzar el problema en el peor caso nos percatamos que podía tratarse de una estrategia sumamente compleja, como se muestra a continuación:

> *Sea S un objeto llamado **estado**, que representa todos los posibles subconjuntos de minutos asignados o no asignados, S estaría definido por:*
>
> $$S = 2^{(900 \text{ minutos/día}) * (5 \text{ días}) * (\text{Espacio físico})}$$
>
> *Sea N el número de UEA que se desean asignar, G el número de grupos por UEA, el peor caso estaría acotado por:*
>
> $$\textit{Peor caso} = N * G * S$$

Razón por la cual, decidimos cambiar la estrategia y se procedió a realizar un algoritmo recursivo que fuera evaluando todas las posibles alternativas, guardando un registro de los minutos asignados bajo esa alternativa para después elegir aquella que maximice el número de minutos asignados.

Semanas 7, 8 [Del 22 de Octubre al 2 de Noviembre de 2012]

Actividades Realizadas:

- Elaboración de las primeras versiones del algoritmo de asignación.
- Creación de la clase *Iterador_asignacion* que busca asignar una sesión dentro de una lista de horarios.

Descripción:

En estas semanas se procedió a realizar las primeras versiones del algoritmo de asignación, el esquema general del algoritmo se presenta en la Figura 1.

```
int asignacion (uea, grupos){
       SI uea es la última ENTONCES
              REGRESA 0
       CASO CONTRARIO
              res= Evaluar asignacion(uea+1,grupos)
              SI grupos >0 ENTONCES
                  MIENTRAS(Se encuentre la manera de asignar un grupo)HACER
                     ben=Calcular el beneficio de haber asignado ese grupo.
                     siguiente= Evaluar asignacion(uea, grupos-1)
                     horas = MAX (res, siguiente+ ben)
                  FIN MIENTRAS
              FIN SI
       FIN SI
       REGRESA horas
}
```

Figura 1. Esquema general del algoritmo de asignación recursivo.

Para la realización de asignaciones, se construyó una clase llamada *Iterador_asignación*, dicha clase está encargada de mantener un apuntador que informe la posición actual dentro de una lista de un espacio físico (aunque en realidad desconoce sobre cual lista), además permite definir en dónde se encuentran ubicados los nodos sesión y preparación, así como la posición dentro del nodo en donde comienza cada uno de ellos. La base principal de esta clase es el método *siguiente_horario*, el cual se encarga de buscar cuál es el siguiente horario disponible en el cual se puede asignar una sesión, considera que el tiempo de preparación puede realizarse de manera interrumpida, sin embargo, las sesiones no pueden hacerlo.

Semanas 9 y 10 [Del 5 de Noviembre al 16 de Noviembre de 2012]

Actividades Realizadas:

- Modificaciones realizadas al algoritmo de asignación.
- Creación de la clase *Restaurador* usada para guardar información de cómo se encontraba un espacio físico antes de haber realizado una asignación.
- Creación de la clase *Pareja* usada para guardar un arreglo de objetos *Restaurador* así como el número de minutos asignados hasta el momento.

Descripción:

En este periodo se realizaron algunas modificaciones al algoritmo de asignación, con el fin de que se pudiera tener una manera de restaurar los datos a una versión previa, ya que el algoritmo evalúa qué tan conveniente es omitir completamente una UEA en relación a intentar asignar un grupo de dicha UEA y llamarse recursivamente con un grupo menos.

Para poder efectuar los cambios realizados en el algoritmo, fue necesario implementar dos clases más. La primera es la clase *Restaurador*, esta clase está encargada de guardar la información necesaria para poder 'regresar' un espacio físico a su estado original, para ello necesita guardar el nodo a partir del cual se modificó el espacio, el espacio original (que fue reemplazado), el iterador de asignación original, así como la información del espacio físico: (Índice y UEA).

Por otra parte, también se creó la clase *Pareja*, la cual consiste simplemente en un arreglo de objetos *Restaurador* así como el número de minutos asignados hasta el momento.

El esquema general del algoritmo se presenta en la Figura 2.

```
Pareja asignacion (uea, grupos) {
       SI uea es la última ENTONCES
              REGRESA Pareja (0,new ArrayList<Restaurador>)
       CASO CONTRARIO
              Pareja res= asignación (uea+1, grupos)
              SI grupos >0 ENTONCES
                  ArrayList<Restaurador> arr= Arreglo con los restauradores de
                  las asignaciones asignadas.
                  MIENTRAS(Se encuentre la manera de asignar un grupo)HACER
                     ben= uea.sesiones * uea.duracion_sesion
                     Pareja siguiente = asignacion(uea, grupos-1)
                     SI (siguiente.horas + ben > res.horas) ENTONCES
                            res.horas = siguiente.horas + beneficio
                            res.restauradores= arr +siguiente.restauradores
                     FIN SI
                  FIN MIENTRAS
              FIN SI
       FIN SI
       REGRESA res
}
```

Figura 2. Esquema general del algoritmo de asignación recursivo.

Semana 11 y 11 Bis [Del 19 al 30 de Noviembre de 2012]

Actividades Realizadas:

- Aplicación de correcciones y pruebas.
- Envío de los resultados finales a PHP
- Elaboración de documentación y reporte final.

Descripción:

Durante esta etapa se realizó el proceso de verificación de cada uno de los métodos y clases que conforman el proyecto, y una vez verificado el correcto funcionamiento del algoritmo, se procedió a enviar los resultados finales a PHP, donde se realizó la interpretación y presentación de los datos en la página web.

Asimismo, se realizaron diversas pruebas con el fin de ejemplificar el funcionamiento correcto del algoritmo en diversas situaciones, verificando que el algoritmo siempre realice las asignaciones buscando maximizar el número de minutos utilizados en los espacios físicos. Algunas de estas pruebas pueden revisarse en el *apéndice C* de este reporte.

Además, durante estas semanas se elaboró la documentación y el reporte del Proyecto Terminal II.

Conclusiones

Gracias a la implementación de este proyecto, podemos concluir que el problema de la asignación óptima para el uso de espacios físicos representa un desafío difícil de resolver, ya que en él intervienen una gran cantidad de variables (como en nuestro caso fueron las UEA, salones, número de grupos, número de sesiones, duración de cada sesión, tiempo de preparación, resolución) y restricciones (como lo fueron los horarios de trabajo del personal), y la estrategia de programación dinámica que inicialmente consideramos utilizar resultó ser inviable para encontrar una buena asignación debido a la complejidad que involucraba el algoritmo, siendo costoso computacionalmente hablando en tiempo y memoria. Por esta razón decidimos elaborar un algoritmo recursivo utilizando la estrategia de fuerza bruta, el cual arroja una solución en un tiempo computacional razonable y utilizando una pequeña cantidad de recursos.

Además, consideramos que el proyecto realizado cumple con los requisitos que se establecieron en la propuesta de proyecto terminal, ya que además de encontrar una solución al problema de la asignación para el uso de laboratorios y talleres de la División de Ciencias Básicas e Ingeniería, representa una forma de automatizar un proceso que actualmente se realiza de forma manual e involucra reunir a diversos representantes de la División con el fin de llevar a cabo la programación trimestral de dichos espacios.

Bibliografía

[1] E. Sordo, "Presupuesto 2011. CBI-A", UAM Azcapotzalco, Distrito Federal, México, Nov. 2010, p. 17.

[2] *Plan de Desarrollo 2010-2013*, UAM Azcapotzalco, Distrito Federal, México, 2010, p. 30.

[3] E. Oliva, "Sistema de asignación de ayudantes a profesores del Departamento de Sistemas de la División de Ciencias Básicas e Ingeniería", Proyecto terminal de Ingeniería en Computación, Universidad Autónoma Metropolitana, D.F, México, 2011.

[4] J. González, "Algoritmo GRASP para la asignación de carga académica a docentes", Proyecto terminal de Ingeniería en Computación, Universidad Autónoma Metropolitana, D.F, México, 2011.

[5] A. Ramírez, "Sistema de gestión de carga académica para el Departamento de Sistemas de CBI", Proyecto terminal de Ingeniería en Computación, Universidad Autónoma Metropolitana, D.F, México, 2011.

[6] M. Durán, (2012, Feb. 19). "Sistema experto para asignación de horarios en la Escuela Superior de Cómputo.", Trabajo terminal TT0176, Escuela Superior de Cómputo, Instituto Politécnico Nacional, México, 2001. [En línea].
Disponible: http://fgalindosoria.com/ligas/tt/#TrabajosTerminalesCuartaGeneración

[7] J. Mejía, (2012, Feb. 20). "Asignación de Horarios de clases universitarias mediante algoritmos evolutivos.", Trabajo de Maestría en Ingeniería Industrial, División de Postgrados e Investigaciones en Ingeniería, Universidad del Norte, Barranquilla, Colombia, 2008. [En línea].
Disponible: http://manglar.uninorte.edu.co/bitstream/10584/80/1/84032706.pdf

Apéndices

Tal y como se señaló en la sección *Especificación Técnica* de la propuesta de Proyecto Terminal, a continuación se presentan los siguientes documentos con el fin de facilitar el modo de empleo del proyecto desarrollado.

Apéndice A. Manual de instalación

Apéndice B. Manual del usuario.

Apéndice C. Ejemplos de funcionamiento.

Apéndice D. Código fuente desarrollado

Apéndice A. Manual de instalación

Para que el algoritmo de *asignación óptima para el uso de laboratorios y talleres de la División de Ciencias Básicas e Ingeniería* pueda funcionar de manera correcta, es necesario que el equipo tenga instalado el siguiente software:

- PHP.

  Debido a que el proyecto se desarrolló utilizando el lenguaje de programación PHP, es necesario que el equipo cuente con dicho software del lado del servidor. Como se mencionó en la propuesta, se buscaba que el sistema no se apegara a ningún tipo de sistema operativo en especial, por lo que se desarrolló específicamente para acceso mediante la web.

- Java

  Debido a que la totalidad del algoritmo se realizó utilizando el lenguaje de programación Java, es necesario que el equipo cuente con dicho software para que el archivo de php pueda mandar y recibir los datos correctamente al jar (*Java ARrchive*), el cual permite ejecutar aplicaciones escritas en el lenguaje Java.

- Apache

  Por último, dado que el archivo de php se encuentra en el mismo equipo, es necesario instalar el servidor web HTTP Apache en su versión 2.

Cabe destacar que una vez que se haya instalado el software que se describió anteriormente, el proyecto no necesita de configuración especial para su correcta ejecución.

Para que el proyecto pueda ejecutarse correctamente, y una vez instalado el software comentado anteriormente, es necesario que se abra a través de un navegador web el archivo denominado "*index.php"*, verificando previamente que en el mismo directorio en el cual se encuentra dicho archivo, también esté ubicado el jar llamado "*Proceso.jar*", ya que el archivo de php está configurado para buscar el jar dentro de su mismo directorio.

Por último, una vez que se ha accedido a la página "*index.php"*, es necesario que se cuente con dos archivos de texto plano, uno que concentre la información de los espacios disponibles, mientras que el otro debe de poseer la información de los datos de demanda. El formato de ambos archivos se explica en el *apéndice B* de este reporte.

Apéndice B. Manual del usuario

Este manual permitirá a los usuarios aprender a invocar el algoritmo de *asignación óptima para el uso de laboratorios y talleres de la División de Ciencias Básicas e Ingeniería*. Debemos destacar la facilidad del proyecto, debido a que para hacer uso de él, sólo se necesita haber creado previamente los archivos que a continuación se presentan.

I. *Estructura de los archivos*

Para la ejecución del proyecto, es necesario ingresar dos archivos de texto plano, uno que concentre la información de los espacios físicos disponibles, mientras que el otro debe de poseer la información con los datos de demanda, es decir la cantidad de UEA que se desean asignar, con el fin de que el sistema sea capaz de leer fácilmente dichos datos y pueda procesar la información dada.

Estructura del archivo ***Espacios Disponibles***

a) La clave de identificación del espacio.

b) *El horario de trabajo de los encargados de dicho espacio*.
Para representarlo se indica un día y un intervalo de tiempo (*inicio* y *fin*). Se considera que los horarios disponibles en un día dado son la unión de los intervalos indicados para ese día.

La primera cadena de texto (*Día*) debe corresponder con los primeros 3 caracteres de algún día hábil de la semana, mientras que la segunda (*Horario inicial*) y tercera cadena de texto (*Horario final*) deben corresponder con el formato HH:MM.

c) El tiempo de preparación en horas necesario para poder hacer uso del espacio.

En la Figura 1 se muestra el formato que contiene dicho archivo.

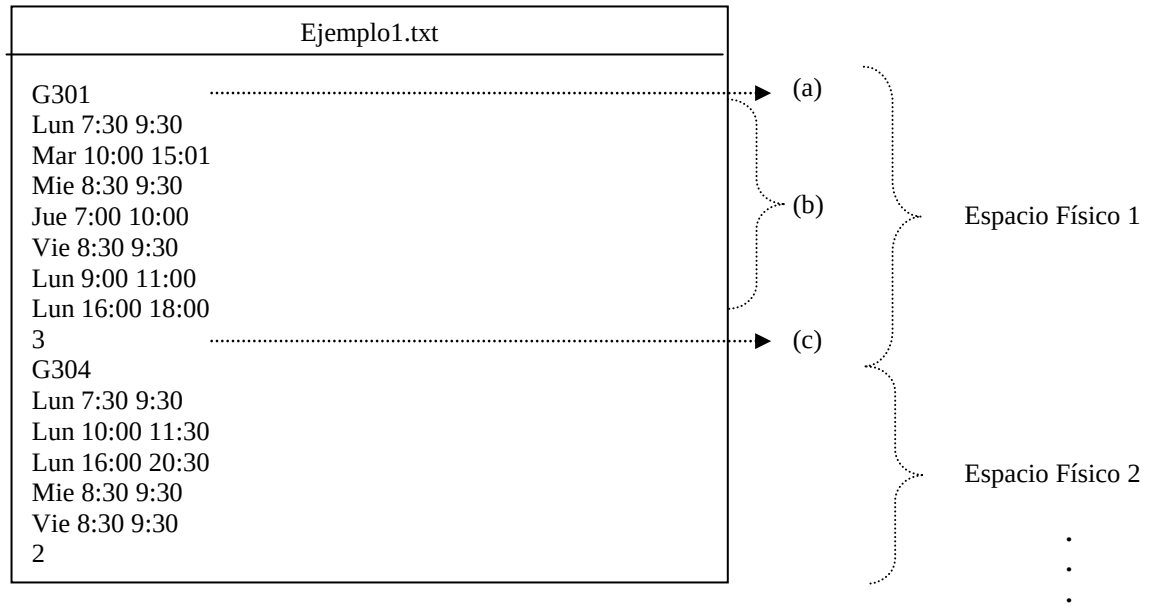

Figura 1. Ejemplo del formato de archivo para la gestión de demanda.

*Estructura del archivo **Demanda***

La información que se debe ingresar en relación a los datos de demanda es:

a) La clave y nombre de una UEA que requiera el uso de un laboratorio o taller.

b) La cantidad de grupos que se requieren abrir de dicha UEA.

c) El número de sesiones a la semana que requiere cada grupo.

d) La duración de cada sesión en horas.

e) Los espacios físicos en los cuales se requeriría impartir dicha UEA.

En la Figura 2 se muestra un ejemplo del formato que contiene dicho archivo:

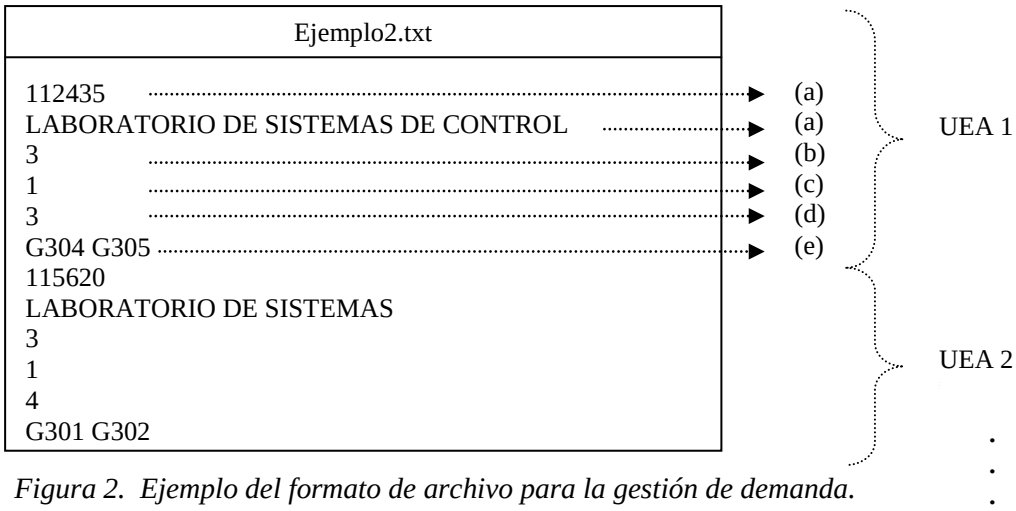

Figura 2. Ejemplo del formato de archivo para la gestión de demanda.

II. Subida de los archivos

Una vez que se hayan generado ambos archivos, se debe acceder a la página "*index.php"*, donde deben de colocarse los archivos como se muestra en la figura 3 (a, b):

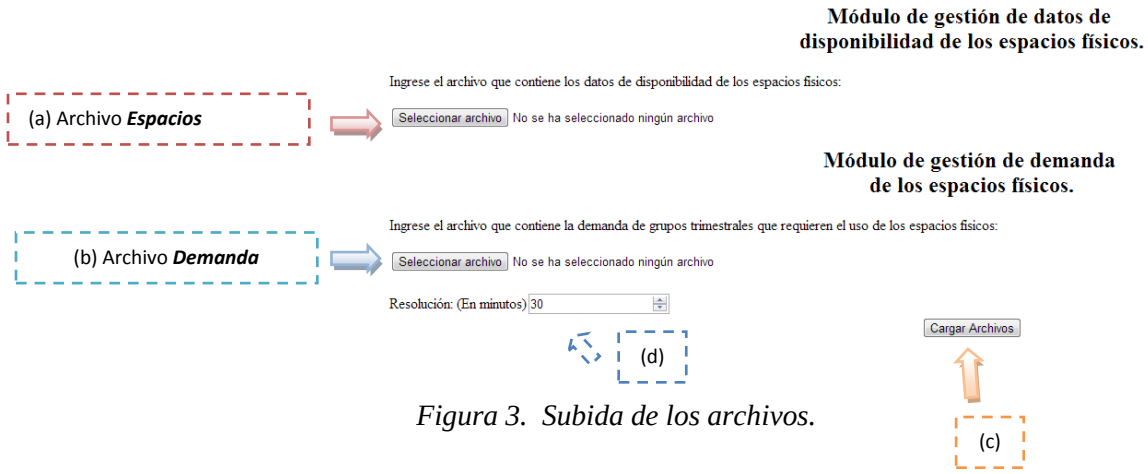

Figura 3. Subida de los archivos.

Posteriormente, se debe seleccionar la **resolución** deseada (intervalo de minutos que el algoritmo debe de separar entre las posibles opciones para una misma asignación de sesión) y se debe hacer click en el botón "*Cargar Archivos*" (figura 3 c,d).

III. Resultados

Finalmente, la página mostrará las UEA de las cuales pudo asignar grupos, indicando los horarios para cada una de las sesiones. (Figura 4)

Resultados

El número total de minutos asignados fueron: **48** minutos
El número total de sesiones asignados fueron: **1**

LABORATORIO DE SISTEMAS DE CONTROL (112435)

Número de grupos deseados: 1

Número de grupos asignados: Ninguno

Características de la UEA:

- Número de sesiones a la semana por grupo: 1
- Duración de cada sesión: 30 minutos

LABORATORIO DE SISTEMAS (115432)

Número de grupos deseados: 1

Número de grupos asignados: 1

Características de la UEA:

- Número de sesiones a la semana por grupo: 1
- Duración de cada sesión: 48 minutos
- Espacios en los que se podía asignar la UEA: G301

Grupos Asignados:

Grupo No. 1
Salón : **G301**
Tiempo de Preparación del salón : **60 minutos**
Preparación : **LUNES a las 7:00 horas - LUNES a las 8:00 horas**
Sesión : **LUNES a las 8:00 horas - LUNES a las 8:48 horas**

Figura 4. Resultados

Apéndice C. Ejemplos de funcionamiento

A continuación se presentan siete ejemplos que contienen distintas características (como son el número de grupos, UEA, sesiones por semana, tiempos de preparación y sesión, resolución, etc.), con el fin de ejemplificar el funcionamiento correcto del algoritmo en diversas circunstancias, y como es el caso del ejemplo 7, el algoritmo siempre realiza las asignaciones buscando maximizar el número de minutos utilizados en los espacios físicos.

Cabe señalar que los archivos de texto (Datos de disponibilidad y de demanda) de cada uno de los ejemplos, se encuentran disponibles en la carpeta "*Ejemplos*", dentro del código fuente de PHP.

Ejemplo 1. Una UEA con 3 sesiones

Resolución: 15 minutos

Unidades de Enseñanza Aprendizaje

No. UEA	No. Grupos	No. Sesiones	Tiempo Sesión Hrs.	Salones
1	1	3	1	G301 G302

Espacios Físicos

No. Salón	Espacios Disponibles	Tiempo de Preparación Hrs.
G301	Lun 7:00 10:00 Lun 11:00 13:00 Lun 14:00 16:00 Lun 17:00 21:10 Lun 21:40 22:00 Mie 7:00 10:00 Mie 11:00 13:00 Mie 14:00 16:00 Mie 17:00 21:10 Mie 21:40 22:00	1
G302	Mar 7:00 8:30 Mar 10:00 11:30 Mar 13:00 15:00 Jue 16:00 17:30 Jue 18:00 19:30 Jue 21:00 22:00	0.5

Resultado. Minutos Asignados: 180/180

LABORATORIO DE SISTEMAS DE CONTROL (112435)

Número de grupos deseados: 1

Número de grupos asignados: 1

Características de la UEA:

- Número de sesiones a la semana por grupo: 3
- Duración de cada sesión: 60 minutos
- Espacios en los que se podía asignar la UEA: G301 G302

Grupos Asignados:

Grupo No. 1

Salón : **G301**
Tiempo de Preparación del salón : **60 minutos**
Preparación : **LUNES a las 7:00 horas - LUNES a las 8:00 horas**
Sesión : **LUNES a las 8:00 horas - LUNES a las 9:00 horas**

Salón : **G301**
Tiempo de Preparación del salón : **60 minutos**
Preparación : **LUNES a las 9:15 horas - LUNES a las 10:15 horas**
Sesión : **LUNES a las 11:15 horas - LUNES a las 12:15 horas**

Salón : **G301**
Tiempo de Preparación del salón : **60 minutos**
Preparación : **LUNES a las 12:30 horas - LUNES a las 13:30 horas**
Sesión : **LUNES a las 14:30 horas - LUNES a las 15:30 horas**

Ejemplo 2. Dos UEA's con 3 y 2 sesiones respectivamente.

Resolución: 15 minutos

Unidades de Enseñanza Aprendizaje

No. UEA	No. Grupos	No. Sesiones	Tiempo Sesión Hrs.	Salones
1	1	3	1	G301 G302
2	1	2	2	G301 G302

Espacios Físicos

No. Salón	Espacios Disponibles	Tiempo de Preparación Hrs.
G301	Lun 7:00 10:00 Lun 11:00 13:00 Lun 14:00 16:00 Lun 17:00 21:10 Lun 21:40 22:00 Mie 7:00 10:00	1.5

	Mie 11:00 13:00 Mie 14:00 16:00 Mie 17:00 21:10 Mie 21:40 22:00	
G302	Mar 7:00 8:30 Mar 10:00 11:30 Mar 13:00 15:00 Jue 16:00 17:30 Jue 18:00 19:30 Jue 21:00 22:00	1

Resultado. Minutos Asignados: 420/420

LABORATORIO DE SISTEMAS DE CONTROL (112435)

Número de grupos deseados: 1

Número de grupos asignados: 1

Características de la UEA:

- Número de sesiones a la semana por grupo: 3
- Duración de cada sesión: 60 minutos
- Espacios en los que se podía asignar la UEA: G301 G302

Grupos Asignados:

Grupo No. 1

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **LUNES a las 7:00 horas - LUNES a las 8:30 horas**
Sesión : **LUNES a las 8:30 horas - LUNES a las 9:30 horas**

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **LUNES a las 11:30 horas - LUNES a las 13:00 horas**
Sesión : **LUNES a las 14:00 horas - LUNES a las 15:00 horas**

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **LUNES a las 15:15 horas - LUNES a las 16:45 horas**
Sesión : **LUNES a las 17:45 horas - LUNES a las 18:45 horas**

LABORATORIO DE SISTEMAS (115432)

Número de grupos deseados: 1

Número de grupos asignados: 1

Características de la UEA:

- Número de sesiones a la semana por grupo: 2
- Duración de cada sesión: 120 minutos
- Espacios en los que se podía asignar la UEA: G301 G302

Grupos Asignados:

Grupo No. 1

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **LUNES a las 9:30 horas - LUNES a las 11:00 horas**
Sesión : **LUNES a las 19:00 horas - LUNES a las 21:00 horas**

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **MIÉRCOLES a las 8:30 horas - MIÉRCOLES a las 10:00 horas**
Sesión : **MIÉRCOLES a las 11:00 horas - MIÉRCOLES a las 13:00 horas**

Ejemplo 3. Dos UEA's con 3 y 2 sesiones respectivamente con más de 1 grupo.

Resolución: 30 minutos

Unidades de Enseñanza Aprendizaje

No. UEA	No. Grupos	No. Sesiones	Tiempo Sesión Hrs.	Salones
1	1	3	1	G301 G302
2	2	2	2	G301 G302

Espacios Físicos

No. Salón	Espacios Disponibles	Tiempo de Preparación Hrs.
G301	Lun 7:00 10:00 Lun 11:00 13:00 Lun 14:00 16:00 Lun 17:00 21:10 Lun 21:40 22:00 Mie 7:00 10:00 Mie 11:00 13:00 Mie 14:00 16:00 Mie 17:00 21:10 Mie 21:40 22:00	1.5
G302	Mar 7:00 8:30 Mar 10:00 11:30 Mar 13:00 15:00 Jue 16:00 17:30 Jue 18:00 19:30 Jue 21:00 22:00	1

Resultado. Minutos Asignados: 660/660

LABORATORIO DE SISTEMAS DE CONTROL (112435)

Número de grupos deseados: 1

Número de grupos asignados: 1

Características de la UEA:

- Número de sesiones a la semana por grupo: 3
- Duración de cada sesión: 60 minutos
- Espacios en los que se podía asignar la UEA: G301 G302

Grupos Asignados:

Grupo No. 1

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **LUNES a las 7:00 horas - LUNES a las 8:30 horas**
Sesión : **LUNES a las 8:30 horas - LUNES a las 9:30 horas**

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **LUNES a las 11:30 horas - LUNES a las 13:00 horas**
Sesión : **LUNES a las 14:00 horas - LUNES a las 15:00 horas**

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **LUNES a las 15:30 horas - LUNES a las 17:00 horas**
Sesión : **LUNES a las 18:00 horas - LUNES a las 19:00 horas**

LABORATORIO DE SISTEMAS (115432)

Número de grupos deseados: 2

Número de grupos asignados: 2

Características de la UEA:

- Número de sesiones a la semana por grupo: 2
- Duración de cada sesión: 120 minutos
- Espacios en los que se podía asignar la UEA: G301 G302

Grupos Asignados:

Grupo No. 1

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **LUNES a las 9:30 horas - LUNES a las 11:00 horas**
Sesión : **LUNES a las 19:00 horas - LUNES a las 21:00 horas**

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **MIÉRCOLES a las 8:30 horas - MIÉRCOLES a las 10:00 horas**
Sesión : **MIÉRCOLES a las 11:00 horas - MIÉRCOLES a las 13:00 horas**

Grupo No. 2

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **MIÉRCOLES a las 7:00 horas - MIÉRCOLES a las 8:30 horas**
Sesión : **MIÉRCOLES a las 14:00 horas - MIÉRCOLES a las 16:00 horas**

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **MIÉRCOLES a las 17:30 horas - MIÉRCOLES a las 19:00 horas**
Sesión : **MIÉRCOLES a las 19:00 horas - MIÉRCOLES a las 21:00 horas**

Ejemplo 4. UEA's de 4 y 3 grupos con más de 1 sesión

Resolución: 30 minutos

Unidades de Enseñanza Aprendizaje

No. UEA	No. Grupos	No. Sesiones	Tiempo Sesión Hrs.	Salones
1	4	2	3	G303 G304
2	3	3	4	G302 G303 G304

Espacios Físicos

No. Salón	Espacios Disponibles	Tiempo de Preparación Hrs.
G301	LUN 7:30 11:00 LUN 16:00 18:00 MAR 10:00 18:00 MIE 8:30 12:00 JUE 7:00 10:00 JUE 15:00 21:00 VIE 8:30 9:30 VIE 10:00 16:00	3
G302	LUN 7:00 9:30 LUN 10:00 11:30 LUN 16:00 20:30 MAR 10:00 18:00 MIE 8:30 9:30 JUE 7:00 10:00 JUE 16:00 21:00 VIE 8:30 9:30	2
G303	LUN 16:00 21:00 MAR 10:00 18:00 MIE 8:30 9:30 JUE 7:00 10:00 JUE 16:00 22:00 VIE 8:30 9:30	3
G304	LUN 7:00 7:02 MAR 9:00 12:00	1

Resultado. Minutos Asignados: 1080/3600

LABORATORIO DE SISTEMAS DE CONTROL (112435)

Número de grupos deseados: 4

Número de grupos asignados: 1

Características de la UEA:

- Número de sesiones a la semana por grupo: 2
- Duración de cada sesión: 180 minutos
- Espacios en los que se podía asignar la UEA: G303 G304

Grupos Asignados:

Grupo No. 1

Salón : **G303**
Tiempo de Preparación del salón : **180 minutos**
Preparación : **LUNES a las 18:00 horas - LUNES a las 21:00 horas**
Sesión : **MARTES a las 10:00 horas - MARTES a las 13:00 horas**

Salón : **G303**
Tiempo de Preparación del salón : **180 minutos**
Preparación : **MARTES a las 16:00 horas - MARTES a las 19:00 horas**
Sesión : **JUEVES a las 7:00 horas - JUEVES a las 10:00 horas**

LABORATORIO DE SISTEMAS (115432)

Número de grupos deseados: 3

Número de grupos asignados: 1

Características de la UEA:

- Número de sesiones a la semana por grupo: 3
- Duración de cada sesión: 240 minutos
- Espacios en los que se podía asignar la UEA: G302 G303 G304

Grupos Asignados:

Grupo No. 1

Salón : **G302**
Tiempo de Preparación del salón : **120 minutos**
Preparación : **LUNES a las 9:00 horas - LUNES a las 11:00 horas**
Sesión : **LUNES a las 16:00 horas - LUNES a las 20:00 horas**

Salón : **G302**
Tiempo de Preparación del salón : **120 minutos**
Preparación : **MARTES a las 10:00 horas - MARTES a las 12:00 horas**
Sesión : **MARTES a las 12:00 horas - MARTES a las 16:00 horas**

Salón : **G302**
Tiempo de Preparación del salón : **120 minutos**
Preparación : **JUEVES a las 8:00 horas - JUEVES a las 10:00 horas**
Sesión : **JUEVES a las 16:00 horas - JUEVES a las 20:00 horas**

Ejemplo 5. Dos UEA's donde conviene asignar todas las sesiones de una y ninguna de la otra.

Resolución: 30 minutos

Unidades de Enseñanza Aprendizaje

No. UEA	No. Grupos	No. Sesiones	Tiempo Sesión Hrs.	Salones
1	3	2	2	G301
2	1	2	3	G301

Espacios Físicos

No. Salón	Espacios Disponibles	Tiempo de Preparación Hrs.
G301	Lun 7:00 10:00 Lun 11:00 13:00 Lun 14:00 16:00 Lun 17:00 21:10 Lun 21:40 22:00 Mie 7:00 10:00 Mie 11:00 13:00 Mie 14:00 16:00 Mie 17:00 21:10 Mie 21:40 22:00	1.5

Resultado. Minutos Asignados: 720/1080

LABORATORIO DE SISTEMAS DE CONTROL (112435)

Número de grupos deseados: 3

Número de grupos asignados: 3

Características de la UEA:

- Número de sesiones a la semana por grupo: 2
- Duración de cada sesión: 120 minutos
- Espacios en los que se podía asignar la UEA: G301

Grupos Asignados:

Grupo No. 1

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **LUNES a las 8:30 horas - LUNES a las 10:00 horas**
Sesión : **LUNES a las 11:00 horas - LUNES a las 13:00 horas**

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **LUNES a las 14:30 horas - LUNES a las 16:00 horas**
Sesión : **LUNES a las 17:00 horas - LUNES a las 19:00 horas**

Grupo No. 2

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **LUNES a las 7:30 horas - LUNES a las 9:00 horas**
Sesión : **LUNES a las 19:00 horas - LUNES a las 21:00 horas**

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **MIÉRCOLES a las 8:30 horas - MIÉRCOLES a las 10:00 horas**
Sesión : **MIÉRCOLES a las 11:00 horas - MIÉRCOLES a las 13:00 horas**

Grupo No. 3

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **MIÉRCOLES a las 7:00 horas - MIÉRCOLES a las 8:30 horas**
Sesión : **MIÉRCOLES a las 14:00 horas - MIÉRCOLES a las 16:00 horas**

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **MIÉRCOLES a las 17:30 horas - MIÉRCOLES a las 19:00 horas**
Sesión : **MIÉRCOLES a las 19:00 horas - MIÉRCOLES a las 21:00 horas**

LABORATORIO DE SISTEMAS (115432)

Número de grupos deseados: 1
Número de grupos asignados: Ninguno

Características de la UEA:

- Número de sesiones a la semana por grupo: 2
- Duración de cada sesión: 180 minutos

Ejemplo 6. Ejemplo en el cual se dispone de la totalidad de tiempo del espacio físico, debiendo asignar en los primeros minutos disponibles.

Resolución: 15 minutos

Unidades de Enseñanza Aprendizaje

No. UEA	No. Grupos	No. Sesiones	Tiempo Sesión Hrs.	Salones
1	1	1	2	G301

Espacios Físicos

No. Salón	Espacios Disponibles	Tiempo de Preparación Hrs.
G301	Lun 7:00 22:00 Mar 7:00 22:00 Mie 7:00 22:00 Jue 7:00 22:00 Vie 7:00 22:00	1.5

Resultado. Minutos Asignados: 120/120

LABORATORIO DE SISTEMAS DE CONTROL (112435)

Número de grupos deseados: 1

Número de grupos asignados: 1

Características de la UEA:

- Número de sesiones a la semana por grupo: 1
- Duración de cada sesión: 120 minutos
- Espacios en los que se podía asignar la UEA: G301

Grupos Asignados:

Grupo No. 1

Salón : **G301**
Tiempo de Preparación del salón : **90 minutos**
Preparación : **LUNES a las 7:00 horas - LUNES a las 8:30 horas**
Sesión : **LUNES a las 8:30 horas - LUNES a las 10:30 horas**

Ejemplo 7. 2 UEA's una sola asignación para verificar que asigne la de mayor tiempo.

Resolución: 15 minutos

Unidades de Enseñanza Aprendizaje

No. UEA	No. Grupos	No. Sesiones	Tiempo Sesión Hrs.	Salones
1	1	1	0.5	G301
2	1	1	0.8	G301

Espacios Físicos

No. Salón	Espacios Disponibles	Tiempo de Preparación Hrs.
G301	Lun 7:00 9:00 Lun 10:00 11:00	1

Resultado. Minutos Asignados: 48/78

LABORATORIO DE SISTEMAS DE CONTROL (112435)

Número de grupos deseados: 1

Número de grupos asignados: Ninguno

Características de la UEA:

- Número de sesiones a la semana por grupo: 1
- Duración de cada sesión: 30 minutos

LABORATORIO DE SISTEMAS (115432)

Número de grupos deseados: 1

Número de grupos asignados: 1

Características de la UEA:

- Número de sesiones a la semana por grupo: 1
- Duración de cada sesión: 48 minutos
- Espacios en los que se podía asignar la UEA: G301

Grupos Asignados:

Grupo No. 1

Salón : **G301**
Tiempo de Preparación del salón : **60 minutos**
Preparación : **LUNES a las 7:00 horas - LUNES a las 8:00 horas**
Sesión : **LUNES a las 8:00 horas - LUNES a las 8:48 horas**

Apéndice D. Código fuente desarrollado

Código Fuente de Java

Paquete algoritmo

Clase Dupla

```
package algoritmo;

import java.util.*;

public class Dupla implements Cloneable {
  /* Esta clase será la encargada de contener los intervalos de tiempo disponibles para cada espacio físico
   * Sus atributos son el inicio y el fin del intervalo, a partir de los cuales se puede calcular su duración*/
  public int inicio;
  public int fin;

  //Constructor
  public Dupla(int inicio, int fin){
    this.inicio = inicio;
    this.fin= fin;
  }

  // Métodos

  /* Este método define la operación 'menor que', recibe la dupla a comparar y devuelve un booleano*/
  public boolean menorQue(Dupla comparacion){
    if (this.inicio < comparacion.inicio){
      return true;
    }

    if (this.inicio > comparacion.inicio){
      return false;
    }

    if(this.fin < comparacion.fin){
      return true;
    }
```

```
    else
    {
      return false;
    }
  }

  /* Este método define el intervalo de tiempo entre el inicio y el fin de la dupla*/
  public int duracion(){
    return this.fin - this.inicio;
  }

  //Getters y Setters
  public int getInicio() {
    return inicio;
  }

  public void setInicio(int inicio) {
    this.inicio = inicio;
  }

  public int getFin() {
    return fin;
  }

  public void setFin(int fin) {
    this.fin = fin;
  }


  // Método que clona un objeto de tipo 'Dupla'
  public Object clone(){
        Object obj=null;
        try{
            obj=super.clone();
        }catch(CloneNotSupportedException ex){
            System.out.println(" No se puede duplicar la Dupla");
        }
        return obj;
    }


}
```

```
//Clase comparador que se encarga de definir cuando una dupla 'a' es mayor, menor o igual a la dupla 'b'
class Comparador implements Comparator<Dupla> {
  public int compare(Dupla a, Dupla b)
  {
      if (a.inicio < b.inicio){
        return -1;//Menor
      }

      if (a.inicio > b.inicio){
        return 1;//Mayor
      }

      if(a.fin < b.fin){
        return -1;
      }
      else if (a.fin > b.fin) {
        return 1;
      }

      return 0;//Igual
  }
}
```

Clase Algoritmo

```
package algoritmo;
import java.io.*;
import java.util.ArrayList;
import listaDobleEnlazada.ListaDoble;
import listaDobleEnlazada.Nodo;


public class Algoritmo {

  public static UEA[] ueas;
  public static Horarios[] espacios;
  public static int resolucion;


  /* Este método está encargado de inicializar y llenar el arreglo que recibirá
   * los datos enviados por PHP, almacenándolos en el arreglo 'datos'.   */
  public static void main(String[] args)throws IOException {

    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    int longitud = Integer.parseInt(in.readLine());

    //Arreglo que almacenara los datos de PHP
      String [] datos = new String [longitud];

      //Captura los datos de PHP
      for (int i=0; i < longitud; i++){
        datos[i]=in.readLine();
        //Elimina los espacios que envia PHP
        if(datos[i].substring(0, 1).equals(" ")){
        datos[i]=datos[i].substring(1, datos[i].length());
      }
      }

      try{
        analiza_arreglo(datos);

        //Llama al algoritmo de asignacion
       Pareja par =asignacion(0);
```

```
     //Comienza el envio de datos a PHP
     //Manda el no. minutos asignados
     System.out.println(par.horas);
     //Manda el no. de sesiones asignadas
     System.out.println(par.restauradores.size());
     //Para  cada sesion, manda el no. de la UEA, el no. del salon, el minuto preparacion y el minuto de sesion
     for (int i=0; i< par.restauradores.size(); i++){
       System.out.println(par.restauradores.get(i).uea);
       System.out.println(ueas[par.restauradores.get(i).uea].salones[par.restauradores.get(i).indice_espacio]);
       System.out.println(par.restauradores.get(i).iterador.minuto_preparacion);
       System.out.println(par.restauradores.get(i).iterador.minuto_sesion);
     }
    } catch (Exception e) {
    e.printStackTrace(System.out);
  }


}


/* Este método recibe el arreglo 'datos' y comienza a interpretar la información, obteniendo
 * la resolución, la información de los espacios físicos y la informacion de las UEA's.   */
public static void analiza_arreglo(String[] datos){
  // Comienza la interpretacion del arreglo
    //Obtiene la resolucion
    resolucion= Integer.parseInt(datos[0]);

    //Obtiene la información de los Espacios
    espacios = new Horarios[Integer.parseInt(datos[1])];

    int puntero =2;
    for (int i=0; i < Integer.parseInt(datos[1]) ;i++){
      int num_espacios=Integer.parseInt(datos[puntero++]);
      espacios[i]=new Horarios();
      for (int h=0; h < num_espacios; h++){
        espacios[i].agregaHorario(h, datos[puntero++]);
      }
      espacios[i].simplifica_duplas();
      espacios[i].setTiempo_preparacion(Integer.parseInt(datos[puntero++]));
      espacios[i].imprime_arreglo();
    }

    //Obtiene la informacion de las UEA's
```

```
    ueas = new UEA[Integer.parseInt(datos[puntero++])];

    for (int i=0; i<ueas.length ; i++){
     ueas[i]=new UEA(Integer.parseInt(datos[puntero++]),Integer.parseInt(datos[puntero++]),Integer.parseInt(datos[puntero++]),In
teger.parseInt(datos[puntero++]),datos[puntero++]);
    }


  }

  /*Método encargado de llamar al algoritmo de asignación mientras más UEA's*/
   public static Pareja asignacion(int uea) {
     if (uea == ueas.length) {
       return new Pareja(0, new ArrayList<Restaurador>());
     }

     return asignacion(uea, ueas[uea].num_grupos);
   }


  /* Método encargado de encontrar cúal es el máximo número de horas que se pueden asignar con los datos de disponibilidad
  * y demanda dados, recibe una UEA y el no. de grupos requeridos, y tras considerar 'omitir' esa UEA o intentar asignarla,
  * devolverá el máximo número de horas que se pueden asignar.*/
   public static Pareja asignacion(int uea, int grupos) {
    Pareja res = asignacion(uea+1);

    if (grupos > 0){
      ArrayList<Restaurador> arr = new ArrayList<Restaurador>();

      while (explora_asignaciones(arr,uea, ueas[uea].sesiones)) {
        int beneficio = ueas[uea].sesiones * ueas[uea].duracion_sesion;

        Pareja siguiente = asignacion(uea, grupos-1);

        if (siguiente.horas + beneficio > res.horas) {
          res.horas = siguiente.horas + beneficio;

          res.restauradores.clear( );
          res.restauradores.addAll(arr);
          res.restauradores.addAll(siguiente.restauradores);
        }
      }
    }
```

```
    return res;
  }

  /* Este método recibe como entrada el arreglo de Restauradores, la UEA y el número de sesiones
   * requeridas. Encuentra cómo asignar un grupo. Si ya se había asignado algún grupo anteriormente,
   * deasigna ese e intenta encontrar otra posibilidad para asignarlo. */
  public static boolean explora_asignaciones(ArrayList<Restaurador> arr, int uea, int sesiones){

    if(arr.size() ==0){
      return calcula_sesiones(arr, uea,sesiones, 0, null);
    }

    while (arr.size() != 0){
      Restaurador ult = arr.get(arr.size()-1);
      Iterador_asignacion iter = restaura(ult);

      arr.remove(arr.size()-1);

      if(calcula_sesiones(arr, uea, sesiones, ult.indice_espacio, iter)){
        return true;
      }
    }

    return false;
  }

  /* Este método intenta asignar todas las sesiones que se requieren para una cierta UEA, recibe el arreglo de Restauradores,
   * la UEA, el número de Sesiones, el número del espacio físico actual y el iterador de asignación. Si el método no logra asignar

   * todas las sesiones requeridas, devolverá 'false' y dejará el arreglo como estaba, en caso contrario devolverá 'true'. */
  public static boolean calcula_sesiones (ArrayList<Restaurador> arr, int uea, int sesiones,  int indice_espacio, Iterador_asign
acion iter){
    int puestos = arr.size();

    while(arr.size()!= sesiones){
      if (iter==null){
        if (indice_espacio == ueas[uea].salones.length){ //no se pudieron asignar todas las sesiones, dejar el arreglo como estab
a originalmente
          while(arr.size() > puestos){
            restaura(arr.get(arr.size()-1));
            arr.remove(arr.size ()-1);
          }
```

```
          return false;
        }

        iter =new Iterador_asignacion(espacios[ueas[uea].salones[indice_espacio]].lista.getCabeza() , espacios[ueas[uea].salones[
indice_espacio]].tiempo_preparacion, ueas[uea].duracion_sesion, resolucion);
      }

      if(iter.siguiente_horario()){ //se pudo asignar la sesion, el iterador sabe dónde pero no sabe de qué lista y para cortarla
 debemos ubicarla
        arr.add(reemplaza(uea, indice_espacio, iter));
      }
      else {
        iter=null;
        ++indice_espacio;
      }
    }
    return true;
  }


  /* El método se encarga de 'actualizar' la disponibilidad del espacio físico una vez que se ha logrado asignar un grupo. Recibe
como
   * entrada la UEA, el no. de espacio y el iterador de asignación. Una vez realizada la actualización del espacio y del iterador
de asignación,
   * crea un objeto 'Restaurador' con el fin de que el algoritmo sea capaz de 'recordar'  cómo se encontraba antes de que se asign
ara
   * dicho grupo. */
  public static Restaurador reemplaza(int uea, int indice_espacio, Iterador_asignacion iterador){
    Iterador_asignacion original = (Iterador_asignacion)iterador.clone();

    // ubicamos la lista y el pivote
    ListaDoble horarios = espacios[ueas[uea].salones[indice_espacio]].lista;
    Nodo pivote = iterador.nodo_preparacion.atras;

    //Calcula cuales nodos debe generar
    int sobrante_derecho =iterador.nodo_sesion.horario.fin -(iterador.minuto_sesion + iterador.duracion_sesion);
    int sobrante_izquierdo= iterador.minuto_preparacion - iterador.nodo_preparacion.horario.inicio;

    Nodo temp = iterador.nodo_sesion.getAdelante();

    ListaDoble cortada = horarios.splice(iterador.nodo_preparacion, iterador.nodo_sesion);
    ListaDoble pegar = new ListaDoble();
```

```
    int cuantos =0;
    Nodo inicio_siguiente =null;

    if (sobrante_derecho >0){
      pegar.insertarEnCabeza( new Dupla (  (iterador.minuto_sesion + iterador.duracion_sesion) ,iterador.nodo_sesion.horario.fin
));
      ++cuantos;

      inicio_siguiente = pegar.getCabeza();
    }

    if (sobrante_izquierdo >0){
      pegar.insertarEnCabeza( new Dupla (iterador.nodo_preparacion.horario.inicio, (iterador.nodo_preparacion.horario.inicio+sobr
ante_izquierdo )));
      ++cuantos;
    }

    if (pivote!=null){
      horarios.merge(pivote, pegar);
    }
    else {
      horarios.merge_front(pegar);
    }

    iterador.nodo_preparacion= (inicio_siguiente == null ? temp : inicio_siguiente);
    iterador.nodo_sesion = iterador.nodo_preparacion;

    if (iterador.nodo_preparacion != null) {
      iterador.minuto_sesion = iterador.nodo_sesion.horario.inicio;
      iterador.minuto_preparacion=iterador.minuto_sesion;
    }

    return new Restaurador(pivote, cortada, cuantos, uea, indice_espacio, original);
  }


  /* Este método está encargado de reestablecer el espacio físico y el iterador de asignación al estado en el que se encontraba or
iginalmente
   * antes de asignar determinada sesión. El método recibe un objeto 'Restaurador'. Una vez que restauró la disponibilidad del esp
acio físico,
   * devuelve el iterador de asignación original.*/
  public static Iterador_asignacion restaura(Restaurador r){
    if (r.quitar > 0){
```

```
      Nodo izq = (r.pivote == null ? espacios[ueas[r.uea].salones[r.indice_espacio]].lista.getCabeza() : r.pivote.adelante);
      Nodo der = izq;

      if (r.quitar == 2){
        der = der.adelante;
      }
      espacios[ueas[r.uea].salones[r.indice_espacio]].lista.splice(izq,der);
    }

    if (r.pivote!=null){
      espacios[ueas[r.uea].salones[r.indice_espacio]].lista.merge(r.pivote, r.cortada);
    }
    else {
      espacios[ueas[r.uea].salones[r.indice_espacio]].lista.merge_front(r.cortada);
    }

    return (Iterador_asignacion)r.iterador.clone( );
  }

  /* Imprime la lista y el iterador actual*/
  public  void imprimeIteradorModificado(ListaDoble horarios, Iterador_asignacion iterador){
    System.out.print("\tLa lista modificada es:");
    horarios.visualizar();

    System.out.println("\tIterador de Asignación modificado:");
    if (iterador.nodo_preparacion!=null){
      System.out.println("\t\tMinuto Preparación: " + iterador.minuto_preparacion+ " (" + iterador.nodo_preparacion.horario.inicio
 +"-" + iterador.nodo_preparacion.horario.fin +")");
    }
    else{
      System.out.println("\t\tEl nodo preparación es nulo, ya recorrió toda la lista");
    }

    if (iterador.nodo_sesion!=null){
      System.out.println("\t\tMinuto Sesión: " +iterador.minuto_sesion + " ("+ iterador.nodo_sesion.horario.inicio +"-
" + iterador.nodo_sesion.horario.fin +")");
    }
    else{
      System.out.println("\t\tEl nodo sesión es nulo, ya recorrió toda la lista");
    }
    }
}
```

Clase Horarios

```
package algoritmo;
import java.util.*;
import listaDobleEnlazada.*;


public class Horarios {

  /* Esta clase almacena una lista doblemente enlazada que contiene los intervalos de tiempo que podemos
   * utilizar para cada espacio físico, así como el tiempo de preparación necesario para poder
   * hacer uso de dicho espacio */
  int tiempo_preparacion;
  ListaDoble lista=new ListaDoble();

  // Setter
  public void setTiempo_preparacion(int tiempo_preparacion) {
    this.tiempo_preparacion = tiempo_preparacion;
  }

  // Imprime en pantalla el tiempo de preparación
  public void imprime_arreglo(){
    //System.out.println("\n\tEl tiempo de preparacion necesario es " +tiempo_preparacion + "\n\n");//**
  }

  /* Esta función va construyendo las duplas a partir de la información obtenida de PHP.
   * Al final, todas las duplas son almacenadas en un ArrayList */
  private ArrayList<Dupla> v = new ArrayList<Dupla>();
  public void agregaHorario(int clave, String linea) {
    String[] tmp=new String [2];
    tmp =linea.split(" ");
    int inicio=Integer.parseInt(tmp[0]);
    int fin=Integer.parseInt(tmp[1]);
    Dupla temp= new Dupla(inicio, fin);
    v.add(temp);

  }
```

```java
/* Este método se encarga de ordenar las duplas y colocarlas en el orden correcto, una vez realizado esto,
 * el método se encargará de 'fusionar' las duplas, es decir, une aquellas duplas que pueden reducirse en una sola
 * con el fin de simplificarlas    */
public void simplifica_duplas(){
  ArrayList<Dupla> simplificado = new ArrayList<Dupla>();

  //Muestra las Duplas ANTES de que se ordenen
  Dupla temp;

  for (int i=0; i < v.size(); i++){
    temp=(Dupla) v.get(i);

  }

  //Ordena las Duplas
  Collections.sort(v, new Comparador( ));

  //Muestra las Duplas Despues de  ordenenarlas
  //System.out.println("\n\tDuplas Ordenadas:");
  for (int i=0; i < v.size(); i++){
    temp=(Dupla) v.get(i);
  }

  //Simplifica las Duplas
  int x=0;
  Dupla a,b;

  //Pregunta si el horario sólo está compuesto por una dupla, caso contrario simplifica
  if(v.size()==1){
    simplificado.add(v.get(0));
  }
  else{
    for (x=0; x <  v.size()-1; x++){
      a= (Dupla) v.get(x);
      b= (Dupla) v.get(x+1);

      if ( b.inicio <= a.fin ){
        // Se deben unir las Duplas
        b.setInicio(a.getInicio());

        if(a.fin > b.fin){
          b.setFin(a.getFin());
        }
```

```
        }
        else
        {
            simplificado.add(a);
        }

        if (x== (v.size()-2)){
          simplificado.add(b);
        }
      }
    }


    v=(ArrayList<Dupla>)simplificado.clone();



    for (int i=(v.size()-1); i >=0; i--){
      lista.insertarEnCabeza((Dupla) v.get(i));
    }
   }

}
```

Clase Iterador_asignacion

```
package algoritmo;
import listaDobleEnlazada.*;

public class Iterador_asignacion implements Cloneable {
    /* Esta clase está encargada de mantener un apuntador que informe cuál es la posición actual en donde estamos iterando en una lista de horarios
     * de un espacio físico (aunque desconoce de cúal lista), además permite definir en dónde se encuentran ubicados los nodos sesión y preparación, así como la posición
     * dentro del nodo en donde comienzan */

    Nodo nodo_preparacion;
    int minuto_preparacion;
    int duracion_preparacion;

    Nodo nodo_sesion;
    int minuto_sesion;
    int duracion_sesion;
    int separacion;


    //Constructor, nodo_preparacion apunta al inicio de la lista y nodo_sesion es nulo
    public Iterador_asignacion(Nodo inicial, int dp, int ds, int s) {
      nodo_preparacion = inicial;
      minuto_preparacion= nodo_preparacion.horario.inicio;

      duracion_preparacion= dp;
      duracion_sesion = ds;

      separacion = s;
    }

    /* Este método se encargar de buscar cuál es el siguiente horario diponible en el cual se puede asignar
     * una sesión, considera que el tiempo de preparación puede realizarse de manera interrumpida, sin embargo,
     * las sesiones no pueden hacerlo. Devuelve 'true' si pudo asignar y deja el iterador
     * en la posición donde se encontró la asignación, de lo contrario devuelve 'false'. */
    public boolean siguiente_horario(){

      if (nodo_sesion != null) {
        // no es la primera vez que se busca un horario, debemos avanzar "separacion"
          int faltan = separacion;
```

```
    while (faltan > 0) {
      // si podemos recorrernos en el nodo preparacion para cumplir la separación, lo hacemos  y ya
      int usar = nodo_preparacion.horario.fin - minuto_preparacion;

      if (usar > faltan) {
        minuto_preparacion += faltan;
        break;
      }
      else {
        //si no podemos recorrernos, entonces avanzamos al siguiente nodo y tomamos en cuenta la separación existente
        // en tiempo entre dos horarios de disponibilidad
        Nodo siguiente = nodo_preparacion.getAdelante( );

        if (siguiente == null) {
          return false;
        }

        faltan -= usar + (siguiente.horario.inicio - nodo_preparacion.horario.fin);

        nodo_preparacion = siguiente;
        minuto_preparacion = nodo_preparacion.horario.inicio;
      }
    }

    nodo_sesion = null;
}

int atras=0;
minuto_sesion = minuto_preparacion;

Nodo probar= nodo_preparacion;

do {
  if(nodo_preparacion == null){
    return false;
  }

  // ignorando la preparacion, se busca un nodo en donde quepa la sesion.
  while (probar.horario.fin - minuto_sesion < duracion_sesion){
    atras += probar.horario.fin - minuto_sesion;
    probar=probar.getAdelante();
```

```
      if (probar == null){
        return false;
      }

      minuto_sesion = probar.horario.inicio;
    }

    //Cuando ya se encontro un nodo en el que cabe la sesion y cabe la preparacion con el tiempo acumulado
    if (atras >= duracion_preparacion ){
      //Libera los primeros nodos que no son necesarios para el tiempo de preparacion
      while(atras - (nodo_preparacion.horario.fin - minuto_preparacion) >= duracion_preparacion){
        atras -= (nodo_preparacion.horario.fin - minuto_preparacion);

        nodo_preparacion= nodo_preparacion.getAdelante();
        minuto_preparacion = nodo_preparacion.horario.inicio;
      }

      minuto_preparacion += (atras - duracion_preparacion);
      nodo_sesion= probar;
    }
    else if((duracion_sesion + duracion_preparacion - atras) <= probar.horario.fin - minuto_sesion){
      //no cabe la preparación en el tiempo atras, pero si recorremos la sesión hacia adelante en el mismo nodo, ya cabe
      minuto_sesion += duracion_preparacion - atras;
      nodo_sesion= probar;
    }
    else {
      //No se puede establecer la sesion en ese nodo, se avanza al siguiente Nodo
      atras += probar.horario.fin - minuto_sesion;

      probar= probar.getAdelante();

      if (probar == null){
        return false;
      }

      minuto_sesion = probar.horario.inicio;
    }
  } while (nodo_sesion == null);

  return true;
}

void imprime_datos(Nodo probar){
```

```
      //Imprime Preparación
      if( nodo_preparacion!=null){
        System.out.println("\t\tAhorita en el iterador: Minuto Preparación: " + minuto_preparacion+ " (" + nodo_preparacion.horari
o.inicio +"-" + nodo_preparacion.horario.fin +")");
      }
      else{
        System.out.println("\t\tAhorita en el iterador: Minuto Preparación: " + minuto_preparacion + " (Nodo Preparación Nulo)");


      }
      //Imprime Sesión
      if( nodo_sesion!=null){
        System.out.println("\t\tAhorita en el iterador: Minuto Sesión: " + minuto_sesion+ " (" + nodo_sesion.horario.inicio +"-
" + nodo_sesion.horario.fin +")");
      }else{
        System.out.println("\t\tAhorita en el iterador: Minuto Sesión: " + minuto_sesion + " (Nodo Sesión Nulo)");
      }

      //Imprime Probar  (Si es que existe en el momento en que la mandamos llamar)
      if( probar!=null){
        System.out.println("\t\tAhorita en el iterador: Nodo Probar:  (" + probar.horario.inicio +"-" + probar.horario.fin +")");
      }
    }

    // Método que clona un objeto de tipo 'Iterador de asignación'
    public Object clone(){
          Iterador_asignacion obj=null;
          try{
              obj=(Iterador_asignacion)super.clone();
          }catch(CloneNotSupportedException ex){
              System.out.println(" No se puede duplicar el Iterador de Asignacion");
          }
          return obj;
      }
}
```

Clase Pareja

```
package algoritmo;

import java.util.ArrayList;

public class Pareja {

  int horas;
  ArrayList<Restaurador> restauradores;

  public Pareja(int horas, ArrayList<Restaurador> arreglo) {
    this.horas = horas;
    this.restauradores = arreglo;
  }

  //Imprime los datos de pareja
  public void imprime_pareja(){
    System.out.println("\tEl numero de minutos es " +  horas);
    if(restauradores!=null){
      System.out.println("\tEl numero de restauradores guardados son " + restauradores.size());
      for (int i=0; i< restauradores.size(); i++){
        System.out.println("\tRestaurador No. " + i);
        restauradores.get(i).visualizar();
      }
      System.out.println("\n");
    }
    else{
      System.out.println("\tEl arreglo de restauradores es nulo\n");
    }

  }

  //Getters y Setters
  public int getHoras() {
    return horas;
  }

  public void setHoras(int horas) {
    this.horas = horas;
  }
```

```
  public ArrayList<Restaurador> getArreglo() {
    return restauradores;
  }

  public void setArreglo(ArrayList<Restaurador> arreglo) {
    this.restauradores = arreglo;
  }

}
```

Clase UEA

```java
package algoritmo;

public class UEA {
  /* Esta clase será la encargada de contener la información de cada una de las UEA's.
   * Sus atributos son el no. de grupos que se requieren, el no. de sesiones requeridas,
   * la duración de cada sesión y los salones en los que se puede cursar dicha UEA   */
  int num_grupos;
  int sesiones;
  int duracion_sesion;
  int salones[];

  //Constructor
  UEA(int num_grupos,int num_sesiones, int duracion_sesion, int num_salones, String salones_completos){
    this.num_grupos=num_grupos;
    this.sesiones=num_sesiones;
    this.duracion_sesion=duracion_sesion;

    salones=new int[num_salones];

    String salones_cadena[]= new String[num_salones];
    salones_cadena=salones_completos.split(" ");

    for (int i=0; i< num_salones; i++){
      salones[i]= Integer.parseInt(salones_cadena[i]);
    }

  }

  /* Este método se encarga de publicar toda la información que se tiene de la UEA */
  public void imprime_uea(){
    System.out.println("\n\nEl numero de grupos de la UEA es " + num_grupos);
    System.out.println("\tEl numero de sesiones de la UEA es " + sesiones);
    System.out.println("\tLa duracion de la UEA es " + duracion_sesion);
    System.out.println("\tLa numero de salones es " +  salones.length);
    for (int i =0; i < salones.length; i++){
      System.out.println("\tEl salon " + i + " es " + salones[i]);
    }
  }

}
```

Clase Restaurador

```
package algoritmo;
import listaDobleEnlazada.*;

public class Restaurador {
    /* Esta clase está encargada de guardar la información necesaria para poder 'regresar' un espacio físico
     * a su estado original, para ello necesita guardar el nodo a partir del cual se modificó el espacio (pivote),
     * el espacio utilizado (lista 'cortada'), el iterador de asignación original (iterador), así como la información
     * del espacio físico: índice, UEA. */

    Nodo pivote;
    ListaDoble cortada;

    int indice_espacio;
    int uea;
    int quitar;

    Iterador_asignacion iterador;


    //Constructor
    public Restaurador(Nodo p, ListaDoble c, int q, int u,int i, Iterador_asignacion it) {
      this.pivote = p;
      this.cortada = c;
      this.quitar = q;
      this.indice_espacio= i;
      this.uea =u;
      this.iterador = it;
    }


    //Visualiza los datos que posee el restaurador
    public void visualizar(){
      System.out.println("\tEl objeto RESTAURADOR de la UEA:"+this.uea +" del salon " + this.indice_espacio+" posee los siguiente
s datos:");
      System.out.println("\t\tElementos que se añadieron a la lista (Sobrantes): " + this.quitar);
      //Verifica que el nodo preparación no haya sido la cabeza
      if(this.pivote!=null){
        System.out.println("\t\tNodo Pivote: (" + this.pivote.horario.inicio +"-" +this.pivote.horario.fin + ")");
      }
      else{
```

```
        System.out.println("\t\tEl nodo preparacion era la cabeza de la lista, por lo tanto, el nodo pivote es nulo.");
      }
      System.out.println("\t\tLista Cortada:");
      this.cortada.visualizar();
      System.out.println("\t\tIterador de Asignacion guardado:");
      System.out.println("\t\t\tMinuto Preparacion: " + iterador.minuto_preparacion+ " (" + iterador.nodo_preparacion.horario.inic
io +"-" + iterador.nodo_preparacion.horario.fin +")");
      System.out.println("\t\t\tMinuto Sesion: " +iterador.minuto_sesion + " ("+ iterador.nodo_sesion.horario.inicio +"-
" + iterador.nodo_sesion.horario.fin +")");


    }

}
```

Paquete listaDobleEnlazada

Clase Nodo

```
package listaDobleEnlazada;
import algoritmo.*;


public class Nodo implements Cloneable  {
  public Dupla horario;
  public Nodo adelante;
  public Nodo atras;

  //Constructores
  public Nodo(Dupla e){
    horario=e;
    adelante =null;
    atras=null;
  }

  public Nodo(Dupla e, Nodo n, Nodo a){
    horario = e;
    adelante =n;
    atras=a;
  }

  //Getters y Setters
  public Dupla getHorario(){
    return horario;
  }

  public Nodo getAdelante(){
    return adelante;
  }

  public Nodo getAtras(){
    return atras;
  }


  public void setAdelante(Nodo enlace) {
    this.adelante = enlace;
```

```
  }


  public void setAtras(Nodo enlace) {
    this.atras = enlace;
  }


  // Método que clona un objeto de tipo 'Nodo'
  public Object clone(){
        Nodo obj=null;
        try{
            obj=(Nodo)super.clone();
        }catch(CloneNotSupportedException ex){
            System.out.println(" No se puede duplicar el Nodo");
        }

        return obj;
    }
}
```

Clase ListaDoble

```
package listaDobleEnlazada;
import algoritmo.*;


public class ListaDoble {
  Nodo cabeza;
  Nodo cola;

  //Constructores
  public ListaDoble (){
    cabeza=null;
    cola=null;
  }

  public ListaDoble (Nodo cabeza){
    this.cabeza=cabeza;

  }

  public ListaDoble (Nodo cabeza, Nodo cola){
    this.cabeza=cabeza;
    this.cola=cola;
  }

  /* Función que 'corta' una lista doble desde el Nodo izquierdo hasta el Nodo derecho,
   * devuelve la lista cortada */
  public ListaDoble splice(Nodo izq, Nodo der){

    if (izq.atras != null){
      izq.atras.adelante=der.adelante;
    }
    else
    {
      cabeza=der.adelante;
    }

    if (der.adelante != null){
      der.adelante.atras=izq.atras;
    }
    else
```

```
  {
    cola=izq.atras;
  }

  izq.atras=null;
  der.adelante=null;

  ListaDoble ld = new ListaDoble(izq,der);
  return ld;
}

/* Función que une dos listas enlazadas, recibe el nodo 'pivote', a partir del cual
 * se insertará la lista 'i'. */
public void merge(Nodo pivote, ListaDoble i ){
  if(!i.vacia( )){
    i.cola.adelante= pivote.adelante;

    if(pivote.adelante==null){
      //Actualiza la cola
      cola= i.cola;
    }
    else {
      pivote.adelante.atras = i.cola;
    }

    pivote.adelante = i.cabeza;
    i.cabeza.atras = pivote;
  }
}

/* Funcion que inserta la lista 'i' al inicio de la lista   */
public void merge_front(ListaDoble i){

  if(this.vacia()){
    this.cabeza= i.cabeza;
    this.cola=i.cola;
    return;
  }


  if(!i.vacia()){
    i.cola.adelante=this.cabeza;
    this.cabeza.atras= i.cola;
```

```
    this.setCabeza(i.cabeza);
  }
}

// Función que inserta una dupla en la cabeza de la lista
public ListaDoble insertarEnCabeza(Dupla entrada)
{
    Nodo nuevo;
    nuevo = new Nodo(entrada);
    nuevo.adelante = cabeza;
    if (cabeza != null ){
      // Si ya había al menos un nodo, se actualiza el enlace atras
      cabeza.atras = nuevo;
    }
    else
    {
      //Este es el primer nodo que se inserta. cabeza=cola=nuevo
      cola=nuevo;
    }
    cabeza = nuevo;
    return this;
}

// Función que visualiza en pantalla la lista doble
public void visualizar(){
  Nodo n = cabeza;
  if(n != null){
    System.out.println("\n\tLa cabeza de la Lista es ("+cabeza.horario.inicio + "," + cabeza.horario.fin + ") ");
    System.out.println("\tLa cola de la Lista es ("+cola.horario.inicio + "," + cola.horario.fin + ") ");
  }
  else
  {
    System.out.println("\n\tLa Lista esta vacia.");

  }
  while(n!=null){
    System.out.println("\t("+n.horario.inicio + "," + n.horario.fin + ")");
    n= n.adelante;
  }
}

// Busca el nodo correspondiente a la dupla 'destino' y devuelve dicho Nodo
public Nodo buscarLista (Dupla destino){
```

```
  Nodo indice;
  for(indice = cabeza; indice !=null; indice =indice.adelante)
    if (indice.horario.inicio == destino.inicio && indice.horario.fin == destino.fin)
      return indice;

  return null;
}

// Busca el nodo siguiente al 'actual' y devuelve dicho Nodo
public Nodo next(Nodo actual){
  return actual.adelante;
}

// Busca el nodo anterior  al 'actual' y devuelve dicho Nodo
public Nodo previous(Nodo actual){
  return actual.atras;
}

// Función que inserta una dupla después del nodo 'anterior'
public ListaDoble insertarDespuesDe(Nodo anterior, Dupla entrada){
  Nodo nuevo;
    nuevo = new Nodo(entrada);
    nuevo.adelante = anterior.adelante;
    if (anterior.adelante !=null){
      anterior.adelante.atras = nuevo;
    }
    else
    {
      //El nuevo nodo a insertar será la cola
      cola=nuevo;
    }
    anterior.adelante = nuevo;
    nuevo.atras = anterior;
    return this;
}

// Verifica si la lista está vacía
public boolean vacia( )
{
  return cabeza == null;
}

//Elimina el nodo. Devuelve el nodo anterior al nodo eliminado
```

```
 public Nodo eliminar (Dupla e)
 {
    Nodo actual;
    boolean encontrado = false;
    actual = cabeza;
    // Búsqueda: actual contendra al nodo a Eliminar
    while ((actual != null) && (!encontrado))
    {
      encontrado=(actual.horario.inicio == e.inicio && actual.horario.fin == e.fin);
        if (!encontrado)
          actual = actual.adelante;
    }
    // Enlace de nodo anterior con el siguiente
    if (actual != null)
    {
      //distingue entre nodo cabecera o el resto de la lista
      if (actual == cabeza)
      {
        cabeza = actual.adelante;
        if (actual.adelante != null)
          actual.adelante.atras = null;
      }
      else if (actual.adelante != null)   // No es el último nodo
      {
        actual.atras.adelante = actual.adelante;
        actual.adelante.atras = actual.atras;
      }
      else{
        // último nodo
        actual.atras.adelante = null;
        cola=actual.atras;
        //actual = null;
      }
    }
  return actual.atras;

  }

//Getters y Setters
   public Nodo getCabeza() {
    return cabeza;
  }
```

```
    public void setCabeza(Nodo cabeza) {
      this.cabeza = cabeza;
    }

    public Nodo getCola() {
      return cola;
    }

    public void setCola(Nodo cola) {
      this.cola = cola;
    }
}
```

Código Fuente de PHP

```php
<?php

    // Establece las variables que verifican la subida correcta de los archivos
    $archivo_a =false;
    $archivo_b =false;
    $error=false;

    //**************************************************************************
    // Procesa el archivo del Módulo de gestión de datos de disponibilidad de los espacios físicos
    //**************************************************************************
    $espacios_totales=array();

    if (isset($_FILES['archivo']) && is_uploaded_file($_FILES['archivo']['tmp_name'])) {
        $archivo_a =true;
        $contenido = file($_FILES['archivo']['tmp_name']);

        // Convierte a Mayusculas y quita los espacios de los extremos.
        foreach ($contenido as $clave => $linea) {
            $contenido[$clave] = strtoupper(trim($linea));
        }

        // Arreglo temporal que contendrá a un espacio físico para ser procesado.
        $espacio= array();

        foreach ($contenido as $clave => $linea){

            if(!es_findeespacio($contenido,$clave, $linea)){
                $espacio[]=$linea;

            }
            else
            {
                $espacio[]=$linea;
                conversion($espacio);

                unset($espacio);
            }
        }

    }
```

```
function conversion($arreglo)
{
        if (!array_key_exists($arreglo[0], $GLOBALS["espacios_totales"] )){

            $dato = array();

            foreach ($arreglo as $clave => $linea){

                // Encuentra la clave de identificación del espacio fisico.
                if ($clave==0)    {
                    $id= $linea;

                    // Cuenta el No. de Horarios disponibles
                    $dato ['num_horarios']=(count($arreglo))-2;

                }

                // Procesa los Horarios Disponibles.
                elseif ($clave!=(count($arreglo))-1)    {

                    $temporal=explode(' ', $linea);
                    $resul_convierte_dias=(convierte_dias ($temporal[0]));
                    $resul_conv_tiempo_1=convierte_tiempo ($temporal[1]);
                    $resul_conv_tiempo_2=convierte_tiempo ($temporal[2]);
                    if($resul_convierte_dias!=-1 && $resul_conv_tiempo_1!=-1  &&$resul_conv_tiempo_2!=-1 &&
                    ($resul_conv_tiempo_2>$resul_conv_tiempo_1)){
                        $adicion = $resul_convierte_dias * 901;
                        $dato[]=  ( $resul_conv_tiempo_1     + $adicion)   . " " . (  $resul_conv_tiempo_2  + $adicion) ;
                    }
                    else
                    {
                        echo "<br><font color='red'> ERROR. El Horario <i>$linea </i>del sal&oacute;n $id posee alg&uacute;n
                        error.</font><br>    ";
                        $GLOBALS["error"]=true;
                    }
                }

                else
                {
                    // Encuentra el tiempo de preparación necesario para poder hacer uso del espacio.
```

```
                    if(is_numeric($linea)){
                              $dato ['tiempo']= ceil($linea*60);
                    }
                    else
                    {
                          echo "<br><font color='red'> ERROR. El tiempo de preparaci&oacute;n <i>$linea </i>del
                          sal&oacute;n $id no es v&aacute;lido.</font><br>    ";
                                $GLOBALS["error"]=true;
                         }
                     }
                }


            $GLOBALS["espacios_totales"][$id]=$dato;
        }
        else
        {
            echo "<br><font color='red'> ERROR. El espacio f&iacute;sico '$arreglo[0]' se encuentra ingresado m&aacute;s de
            una vez en el archivo '" . $_FILES['archivo']['name']. "'</font><br>    ";
            $GLOBALS["error"]=true;

        }

}


function convierte_tiempo ($tiempo)
{
    $cadena = explode (':', $tiempo);
    $hora= $cadena[0];
    $minutos = $cadena[1];
    if(is_numeric($hora) && is_numeric($minutos)){
          return ((($hora-7)*60)+ $minutos);
    }
    return -1;

}

function convierte_dias ($dia)
{
    $val=0;
    switch ($dia) {
        case "LUN":
```

```
            $val=0;
            break;
        case "MAR":
            $val=1;
            break;
        case "MIE":
            $val=2;
            break;
        case "JUE":
            $val=3;
            break;
        case "VIE":
            $val=4;
            break;
        default:
            $val=-1;
            break;

    }
    return $val;
}


function total_lineas($arreglo){
    $total=1;
    foreach ($arreglo as $clave => $linea){
            foreach ($arreglo[$clave] as  $linea2){
                $total++;
            }
    }
    return $total;
 }


function es_findeespacio($arreglo, $clave, $linea){

    $cadena=substr ($linea,0,3);

    if ($clave==0 ||$cadena=="LUN" ||$cadena=="MAR" ||$cadena=="MIE" ||$cadena=="JUE" ||$cadena=="VIE" ){

        return false;
    }
    else
```

```
        {
            $cadena=substr ($arreglo[($clave-1)],0,3);
            // Pregunta por el padre.
            if ($cadena=="LUN" ||$cadena=="MAR" ||$cadena=="MIE" ||$cadena=="JUE" ||$cadena=="VIE" ){
              return true;
            }
            else
            {
                return false;
            }
        }

    }


    //*****************************************************************************
    // Procesa el archivo del Módulo de gestión de demanda de los espacios físicos.
//*****************************************************************************
    $demandas_totales=array();
    if (isset($_FILES['archivo2']) && is_uploaded_file($_FILES['archivo2']['tmp_name'])) {
        $archivo_b =true;
        $contenido = file($_FILES['archivo2']['tmp_name']);

        // Convierte a Mayusculas y quita los espacios de los extremos.
        foreach ($contenido as $clave => $linea) {
            $contenido[$clave] = strtoupper(trim($linea));
        }

        // Arreglo Temporal que contendrá la demanda
        $demanda= array();


        $contador=0;
        foreach ($contenido as $clave => $linea){

            if($contador!=5){
                $demanda[]=$linea;
                $contador++;

            }
            else
            {
                $demanda[]=$linea;
```

```
            conversion_gest_demanda($demanda);
            unset($demanda);
            $contador=0;
        }
    }

}

function conversion_gest_demanda($arreglo){
    if (!array_key_exists($arreglo[0], $GLOBALS["demandas_totales"] )){
        $temporal_demanda=array();

        foreach ($arreglo as $clave => $linea){

            switch ($clave) {
                case 0:
                    // Encuentra la clave de la UEA.
                    $id= $linea;
                    break;
                case 1:
                    // Encuentra el nombre de la UEA.
                    $temporal_demanda ['nombre_uea']= $linea;
                    break;
                case 2:
                    // Encuentra la cantidad de grupos que se requieren abrir de dicha UEA
                    if (is_numeric($linea)){
                        $temporal_demanda ['no_grupos']= $linea;
                    }
                    else
                    {
                        echo "<br><font color='red'> ERROR. La cantidad de grupos <i>$linea </i>de la UEA $id no es un dato
                        v&aacute;lido.</font><br>    ";
                        $GLOBALS["error"]=true;
                    }
                    break;
                case 3:
                    // Encuentra el numero de sesiones a la semana que requiere cada grupo.
                    if (is_numeric($linea)){
                        $temporal_demanda ['no_sesiones']= $linea;
                    }
                    else
                    {
                         echo "<br><font color='red'> ERROR. El n&uacute;mero de sesiones <i>$linea </i>de la UEA $id no es un
```

```
                    dato v&aacute;lido.</font><br>";
                    $GLOBALS["error"]=true;
                }
                break;
            case 4:
                // Encuentra la duracion de cada sesion en horas.
                if (is_numeric($linea)){
                    $temporal_demanda ['duracion_sesion']= ceil ($linea*60);
                }
                else
                {
                     echo "<br><font color='red'> ERROR. La duraci&oacute;n <i>$linea </i>de la UEA $id no es un dato
                    v&aacute;lido.</font><br>";
                    $GLOBALS["error"]=true;
                }
                break;
            case 5:
                // Encuentra los espacios fisicos en los cuales se requeriria impartir dicha UEA.
                $temporal_demanda['no_espacios_disponibles']=count(explode(' ', $linea));
                $encuentra_linea=encuentra_id($linea);
                $contiene_id_valido=strpos($encuentra_linea, "-1");

                if($contiene_id_valido==false)
                {
                    $temporal_demanda['espacios_disponibles']=$encuentra_linea;
                }
                else
                {
                    echo "<br><font color='red'> ERROR. No existe un sal&oacute;n de la l&iacute;nea  <i>$linea </i>de la
                    UEA $id.</font><br>";
                    $GLOBALS["error"]=true;
                }
                break;

        }
    }
    $GLOBALS['demandas_totales'][$id]=$temporal_demanda;
}
else
{
    echo "<br><font color='red'> ERROR. La UEA '$arreglo[0] - $arreglo[1]' se encuentra ingresada m&aacute;s de una vez en
    el archivo '" . $_FILES['archivo2']['name']. "'</font><br>    ";
```

```
            $GLOBALS["error"]=true;
        }

    }


    function encuentra_id($linea){
        $tmp=explode(' ', $linea);
        $claves=array_keys($GLOBALS['espacios_totales']);
        $esp="";
        foreach ($tmp as $key => $line){
            $a= array_search($line,$claves);
            if( is_bool($a) ){
                $a=-1;
            }
            $esp = $esp . " ". $a;
        }
        return $esp;
}

    function convierte_num_dia($numero){
        //Encuentra el dia
        if ($numero>=0 && $numero<= 900){
            $dia= "LUNES";
        }
        elseif ($numero>=901 && $numero<= 1801){
            $dia= "MARTES";
        }
        elseif ($numero>=1802 && $numero<= 2702){
            $dia= "MI&Eacute;RCOLES";
        }
        elseif ($numero>=2703 && $numero<= 3603){
            $dia= "JUEVES";
        }
        elseif ($numero>=3604 && $numero<= 4504){
            $dia= "VIERNES";
        }
        else{
            $dia= "DESCONOCIDO";
        }

        return $dia;
    }
```

```
    function convierte_num_hora($numero, $dia){
        //Encuentra la hora y minuto
        switch ($dia){
case "LUNES":
                    $id=0;
                    break;
            case "MARTES":
                    $id=1;
                    break;
            case "MI&Eacute;RCOLES":
                    $id=2;
                    break;
            case "JUEVES":
                    $id=3;
                    break;
            case "VIERNES":
                    $id=4;
                    break;
            default:
                    $id=-1;
                    break;
        }

        $a = $numero - (901*$id);
        $f=  floor($a/60);
        $horas_f= $f + 7;
        $minutos_f = $a - (60*$f);
        if ( strlen($minutos_f)==1){
            $minutos_f= "0". $minutos_f;
        }
        $hora_final = "$horas_f:$minutos_f";
        return $hora_final;

    }


    function convierte_minuto_fecha($cadena){
        $dia =  convierte_num_dia(  $cadena);
        $hora = convierte_num_hora($cadena, $dia);
        $minuto_sesion=$dia ." a las "  .$hora. " horas";
        return $minuto_sesion;
    }
```

```
function busca_espacio_por_posicion($numero){
        $i=0;
        foreach ($GLOBALS["espacios_totales"] as $clave => $linea){
        if ($i==$numero){
                 return $clave;
        }
        else
        {
                $i++;
        }

    }

}

function busca_uea_por_posicion($numero){
   $i=0;
    foreach ($GLOBALS["demandas_totales"] as $clave => $linea){
        if ($i==$numero){
                return $clave;
        }
        else
        {
                $i++;
        }
    }
}


function encuentra_id_salon($linea){
    $salones="";
    for($i=0; $i <sizeof($linea); $i++){
        if($linea[$i]!=""){
            $salones = $salones . " " . busca_espacio_por_posicion($linea[$i]);
        }
    }
    return $salones;
}
```

```
//*****************************************************************************
// Envía ambos archivos a Java
//*****************************************************************************

if ($archivo_a && $archivo_b && !$GLOBALS["error"]) {
    // Inicia el envío a Java
   $descripcion = array(
        0 => array('pipe', 'r'),
        1 => array('pipe', 'w')
    );

    $opciones = array(
        'bypass_shell' => true
   );

    $tuberias = array( );
    $directorio = getcwd( );

    $proceso = proc_open("java -Xincgc -Xmx1G -jar \"$directorio\Proceso.jar\"", $descripcion, $tuberias, null, null,
    $opciones);


    if ($proceso) {

        $total_lineas= total_lineas($GLOBALS["espacios_totales"]) + total_lineas($GLOBALS["demandas_totales"]) -
        count($GLOBALS["demandas_totales"]) +1;

        // Envia el No. de lineas del arreglo
        fwrite($tuberias[0],  $total_lineas . "\n" );


        // Inicia el Envio del Arreglo 1.

        // Envia la Resolucion
        fwrite($tuberias[0],  $_POST['resolucion'] . "\n" );

        // Envia el No. de Espacios Físicos
        fwrite($tuberias[0],  count($GLOBALS["espacios_totales"]) . "\n" );

        // Envía el Arreglo 1
        foreach ($GLOBALS["espacios_totales"] as $clave => $linea){
            foreach ($GLOBALS["espacios_totales"][$clave] as  $linea2){
                fwrite($tuberias[0],  $linea2 . "\n" );
```

```
    }
}

// Inicia el Envio del Arreglo 2.

// Envia el No. de Demandas
fwrite($tuberias[0],  count($GLOBALS["demandas_totales"]) . "\n" );

// Envía el Arreglo 2
foreach ($GLOBALS["demandas_totales"] as $clave => $linea){
  foreach ($GLOBALS["demandas_totales"][$clave] as $key =>  $linea2){
         if($key !="nombre_uea"){
            fwrite($tuberias[0],  $linea2 . "\n" );
        }
    }
}

// Se cierra la tubería
fclose($tuberias[0]);

//Resultados

$minutos_asignados =fgets($tuberias[1]);
$sesiones_totales=fgets($tuberias[1]);

$datos_finales = array();
for ($i=0; $i<$sesiones_totales; $i++){
        $temp= array();
        $temp["num_uea"]=fgets($tuberias[1]);
        $temp["num_salon"]=fgets($tuberias[1]);
        $temp["preparacion"]=fgets($tuberias[1]);
        $temp["sesion"]=fgets($tuberias[1]);
        $datos_finales[$i] = $temp;
}

echo "<h1><center>Resultados </center></h1><br>";
echo "El n&uacute;mero total de minutos asignados fueron: <b>$minutos_asignados </b> minutos<br>";
echo "El n&uacute;mero total de sesiones asignados fueron: <b> $sesiones_totales</b><br>";

//Marca el arreglo de demandas
foreach ($GLOBALS["demandas_totales"] as $clave => $linea){
        $GLOBALS["demandas_totales"] [$clave]["existen_sesiones"]=false;
}
```

```
for ($i=0; $i<$sesiones_totales; $i++){

        //Despliega las sesiones asignadas con sus caracteristicas
        //id_uea
        $clave=    busca_uea_por_posicion($datos_finales[$i]["num_uea"]);

        //salon
        $salon=busca_espacio_por_posicion($datos_finales[$i]["num_salon"]);

        //Inserta al arreglo de UEA's las sesiones asignadas correspondientes
        $GLOBALS["demandas_totales"][$clave]["sesiones_asignadas"][]=array(
        $salon,$datos_finales[$i]["preparacion"],$datos_finales[$i]["sesion"]) ;
        $GLOBALS["demandas_totales"][$clave]["existen_sesiones"]=true;
}


//Muestra la cant. de grupos asignados por UEA
foreach ($GLOBALS["demandas_totales"] as $clave => $linea){

        echo "<hr><h3><center> ". $GLOBALS["demandas_totales"][$clave]["nombre_uea"]." (". $clave.")</h3></center>";
        echo "<h4>N&uacute;mero de grupos deseados: <b>". $GLOBALS["demandas_totales"][$clave]["no_grupos"]
        ."</b><br></h4>";

        if ($GLOBALS["demandas_totales"][$clave]["existen_sesiones"]== true){
         $grupos_asignados =
         (sizeof($GLOBALS["demandas_totales"][$clave]["sesiones_asignadas"]))/($GLOBALS["demandas_totales"][$clave]["
         no_sesiones"]);
         echo "<h4><font color='#FF0000'>N&uacute;mero de grupos asignados: ".
         $grupos_asignados."</h4><br></font>";
          echo "<i>Caracter&iacute;sticas de la UEA:</i>";
          echo "<ul><li>N&uacute;mero de sesiones a la semana por grupo: ".
         $GLOBALS["demandas_totales"][$clave]["no_sesiones"] . "</li>";
         echo "<li>Duraci&oacute;n de cada sesi&oacute;n: ".
         $GLOBALS["demandas_totales"][$clave]["duracion_sesion"] . " minutos</li>";
         $temporal=explode(' ', $GLOBALS["demandas_totales"][$clave]["espacios_disponibles"]);
         echo "<li>Espacios en los que se pod&iacute;a asignar la UEA: ".encuentra_id_salon($temporal)  . "
         </li></ul>";

         echo "<i>Grupos Asignados:</i>";
         $j=0;
         $contador_grupos=0;
```

```
            for($i=0; $i< (sizeof($GLOBALS["demandas_totales"][$clave]["sesiones_asignadas"])); $i++){

                if (($j % ($GLOBALS["demandas_totales"][$clave]["no_sesiones"]))==0 ){
                    echo "<br><dl><dt><b>Grupo No. ".(++$contador_grupos)."</b><br></dt>";
                }

                echo "<dd>Sal&oacute;n :
                <b>".$GLOBALS["demandas_totales"][$clave]["sesiones_asignadas"][$i][0]."</b></dd>";

                $tiempo_preparacion=$GLOBALS["espacios_totales"][$GLOBALS["demandas_totales"][$clave]["sesiones_asign
                adas"][$i][0]]["tiempo"];
                echo "<dd>Tiempo de Preparaci&oacute;n del sal&oacute;n : <b>". $tiempo_preparacion."
                minutos</b></dd>";
                echo "<dd>Preparaci&oacute;n :
                <b>".convierte_minuto_fecha($GLOBALS["demandas_totales"][$clave]["sesiones_asignadas"][$i][1])." -
                ".convierte_minuto_fecha(($GLOBALS["demandas_totales"][$clave]["sesiones_asignadas"][$i][1])+
                $tiempo_preparacion) ."</b></dd>";
                echo "<dd>Sesi&oacute;n :
                <b>".convierte_minuto_fecha($GLOBALS["demandas_totales"][$clave]["sesiones_asignadas"][$i][2])." -
                ".convierte_minuto_fecha(($GLOBALS["demandas_totales"][$clave]["sesiones_asignadas"][$i][2])+$GLOBALS
                ["demandas_totales"][$clave]["duracion_sesion"])."</b></dd>";

                if (($j % ($GLOBALS["demandas_totales"][$clave]["no_sesiones"]))==0 ){
                    echo "</dl>";
                }
                $j++;
            }
        }
        else{
            echo "<h4><font color='#FF0000'>N&uacute;mero de grupos asignados:  Ninguno </h4><br></font>";
            echo "<i>Caracter&iacute;sticas de la UEA:</i>";
            echo "<ul><li>N&uacute;mero de sesiones a la semana por grupo: ".
            $GLOBALS["demandas_totales"][$clave]["no_sesiones"] . "</li>";
            echo "<li>Duraci&oacute;n de cada sesi&oacute;n: ".
            $GLOBALS["demandas_totales"][$clave]["duracion_sesion"] . " minutos</li></ul>";
        }


}

exit;
```

```
            // Se cierra la tubería
            fclose($tuberias[1]);
            proc_close($proceso);
         }
    }


?>
<!Doctype html>
<html>
<body>
       <br>

       <h2><center>M&oacute;dulo de gesti&oacute;n de datos de  </br>disponibilidad de los espacios f&iacute;sicos.</center></h2>

       <p>Ingrese el archivo que contiene los datos de disponibilidad de los espacios f&iacute;sicos:
       </p>

       <form method="post" enctype="multipart/form-data">
              <input type="file" name="archivo">
              <br>

              <h2><center>M&oacute;dulo de gesti&oacute;n de demanda </br>de los espacios f&iacute;sicos.</center></h2>

              <p>Ingrese el archivo que contiene la demanda de grupos trimestrales que requieren el uso de los espacios
              f&iacute;sicos:</p>
              <input type="file" name="archivo2">
              <br><br>
                     Resoluci&oacute;n: (En minutos)<input type="number" name="resolucion" min="1" value="30">
              <center><input type="submit" value="Cargar Archivos"></center>
       </form>
</body>
</html>
```