Departamento de Informática
Universidad Carlos III de Madrid

Grado en Informática

Heurística y Optimización

19 de Enero de 2018

Normas generales del examen

① El tiempo para realizar el examen es de **4 horas**

② No se responderá a ninguna pregunta sobre el examen transcurridos los primeros **30 minutos**

③ Cada pregunta debe responderse en páginas separadas en el mismo orden de sus apartados. Si no se responde, se debe entregar una página en blanco

④ Numera todas las páginas de cada ejercicio separadamente

⑤ Escribe con claridad y en limpio, de forma ordenada y concisa

⑥ Si se sale del aula, no se podrá volver a entrar durante el examen

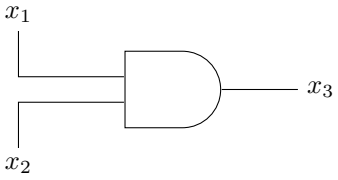
⑦ No se puede presentar el examen escrito a lápiz

Pregunta 1 (1½ puntos)

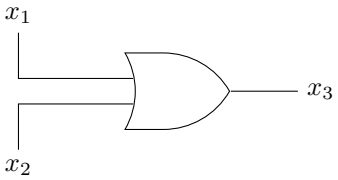
Las puertas lógicas AND y OR implementan las funciones lógicas $\wedge$ y $\vee$, respectivamente, y se pueden emplear, entre otras, para el diseño de circuitos digitales. Se ha decidido emplear *Programación Lineal* para asistir en el diseño automático de circuitos digitales que empleen estas puertas.

Se pide responder razonadamente las siguientes preguntas:

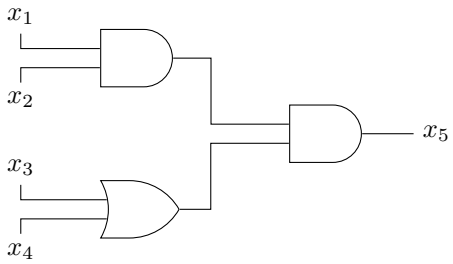
(a) **(½ puntos)** Se pide crear una tarea de Programación Lineal que modelice el comportamiento de una puerta lógica AND, y que determine automáticamente el valor de las entradas x_1 y x_2 para que la salida, x_3 esté activada.



(b) **(½ puntos)** Se pide crear una tarea de Programación Lineal que modelice el comportamiento de una puerta lógica OR, y que determine automáticamente el valor de las entradas x_1 y x_2 para que la salida, x_3 esté desactivada.



(c) **(½ puntos)** Por último, se desea modelizar el circuito que se muestra a continuación:



para determinar el número mínimo de entradas (x_1, x_2, x_3 y x_4) que deben activarse para que se active la salida, x_5.

Pregunta 2 (1½ puntos)

Se dispone de tres productos químicos A, B y C, de los que se sabe que B no puede combinarse ni con A, ni con C. Sin embargo, la combinación de las cantidades necesarias de A y C a la temperatura adecuada da lugar a otro producto, D. Este último producto no debe combinarse nunca ni con A, ni C, pero combinado con B produce otro, E.

Inicialmente no se dispone de ninguno de los productos D o E, pero existen cantidades suficientes de los tres primeros para generarlos.

Se pide responder razonadamente las siguientes preguntas:

(a) **(½ punto)** Modelizar este caso como un problema de *satisfacción de restricciones.*
Es imprenscindible indicar claramente el próposito de cada variable y decisión de diseño tomada.

(b) **(½ puntos)** Determinar si las variables que relacionan los productos A y C son o no *arco-consistentes.* ¿Si o no? ¿Por qué? ¿Ha sido preciso hacer alguna modificación para que lo sean?

(c) **(½ puntos)** Determinar si las variables que relacionan los productos B y C son o no *camino-consistentes* respecto de la variable que representa el producto E. ¿Si o no? ¿Por qué? ¿Ha sido preciso hacer alguna modificación para que lo sean?

Pregunta 3 (3½ puntos)

Considerése el siguiente problema de Programación Lineal:

$$\begin{array}{rcrcr} & & \text{máx}\, z = 3x_1 - 3x_2 & & \\ x_1 & + & x_2 & \geq & 6 \\ x_1 & - & 3x_2 & \geq & -10 \\ x_1 & - & x_2 & \leq & 2 \\ & & \mathbf{x} \geq \mathbf{0} & & \end{array}$$

Se pide responder razonadamente las siguientes preguntas:

(a) **(½ puntos)** Resolver el problema utilizando el método de *resolución gráfica* indicando la solución óptima y el valor de la función objetivo para la solución hallada.

(b) **(½ puntos)** ¿Qué modificaciones es preciso hacer en la tarea de Programación Lineal anterior para hacer que el valor absoluto de la suma de x_1 y x_2 sea menor o igual que 5? Esto es, $|x_1 + x_2| \leq 5$

Considérese ahora, en su lugar, la tarea de Programación Lineal original, con la función objetivo máx $z = 2x_1 - 6x_2$:

$$\begin{array}{rcrcr} & & \text{máx}\, z = 2x_1 - 6x_2 & & \\ x_1 & + & x_2 & \geq & 6 \\ x_1 & - & 3x_2 & \geq & -10 \\ x_1 & - & x_2 & \leq & 2 \\ & & \mathbf{x} \geq \mathbf{0} & & \end{array}$$

y responder razonadamente las siguientes preguntas:

(c) **(½ puntos)** Expresar el problema de Programación Lineal anterior en forma *estándar* de maximización.

(d) **(½ puntos)** Preparar el problema de Programación Lineal que ha resultado del apartado anterior para su aplicación con el algoritmo SIMPLEX para garantizar que pueda comenzarse con la matriz identidad.

(e) **(1 punto)** Resolver el problema de Programación Lineal obtenido en el apartado anterior con el algoritmo SIMPLEX.

Es imprenscindible indicar claramente, en cada iteración: las variables escogidas en la base, su valor, y el valor de la función objetivo

(f) **(½ puntos)** Interpretar las soluciones halladas y explicar qué conclusiones pueden extraerse.

Pregunta 4 (3½ puntos)

Se desea gestionar automáticamente un ascensor inteligente que se encuentra en un edificio con un número arbitrario de plantas n, $n > 2$, y que tiene una capacidad máxima de 10 personas. En cada planta hay usuarios, para cada uno de los cuales se sabe la planta en la que se encuentran, y aquella a la que desean acceder. Una vez que una persona entra en el ascensor, ya no debe salir hasta que haya llegado a su planta de destino. Inicialmente, el ascensor se encuentra en la planta baja, y no importa donde acabe cuando ha atendido todas las peticiones.

Se pide responder razonadamente las siguientes preguntas:

(a) **(½ puntos)** Representar el problema como un *espacio de estados*

(b) **(½ puntos)** ¿Cuál es el máximo factor de ramificación?

(c) **(½ puntos)** Si se desea minimizar el recorrido total que debe hacer el ascensor para atender a todas las personas, ¿qué algoritmo de búsqueda *no informada* sugerirías para su resolución?

(d) **(½ puntos)** Sabiendo que el ascensor tarda sólo 3 segundos en llegar desde una planta a otra inmediatamente adyacente y, por otra parte, la carga y descarga de personas dura siempre 30 segundos, ¿qué algoritmo de búsqueda *no informada* sugerirías para minimizar el tiempo total de operación para atender todas las peticiones?

(e) **(1 punto)** Sugiere una función heurística $h(\cdot)$ que sea *admisible* e *informada* para el problema de minimizar la distancia total recorrida por el ascensor para atender todas las peticiones

(f) **(½ puntos)** Suponiendo que se dispone de una función heurística $h(\cdot)$ admisible para estimar la distancia total que aún debe recorrer el ascensor, ¿qué algoritmo de búsqueda *heurística* es el más indicado para resolver óptimamente el problema? ¿Por qué?

Soluciones del examen de Heurística y Optimización Enero 2018

Problema 1

El problema de modelizar circuitos digitales con tareas de *Programación Lineal* es bien conocido en la literatura científica. Los dos primeros apartados sirven para practicar las dos relaciones lógicas por excelencia con *Programación Lineal* y, el último, cómo combinarlos para modelizar circuitos con el propósito de resolver una cuestión específica. Es importante observar que, en todos los casos, debe modelizarse también la salida producida por cada circuito.

Para la resolución de los tres apartados se seguirá el mismo orden para el diseño de una tarea de *Programación Lineal*:

1. Definición de variables
2. Determinación de las restricciones
3. Diseño de la función objetivo

1. Las variables necesarias para representar el comportamiento del circuito mostrado en el primer apartado serán las que determinan el estado de las dos entradas (x_1 y x_2), y la salida, x_3. Puesto que tanto las entradas como las salidas sólo admiten dos estados (activada o desactivada), todas las variables deben ser binarias, con el 0 representando el estado desactivada, y con 1, el estado activada.

   El propósito de este ejercicio consistía en usar restricciones para crear una región factible que simulara exactamente el comportamiento de una puerta lógica AND. Es decir, que el valor de x_3 fuera 0 si x_1 o x_2 valían 0, y 1 en caso contrario, tal y como se muestra en la Figura 1.

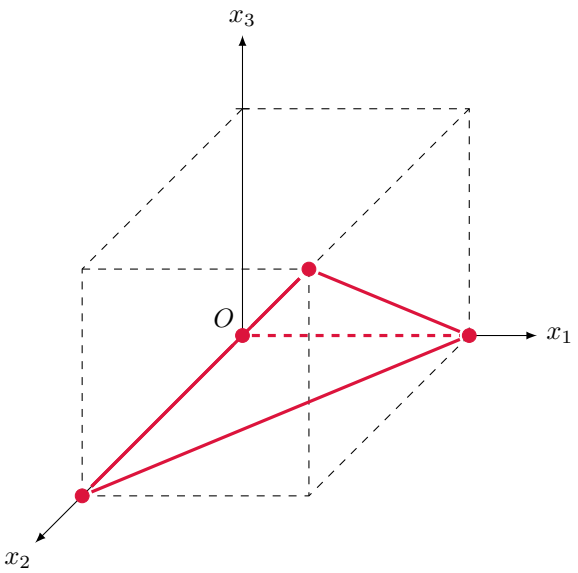


Figura 1: Región factible que simula el comportamiento de una puerta lógica AND

La puerta lógica AND activará la salida sólo cuando sus dos entradas estén activas: $x_3 = x_1 \wedge x_2$. Sumando las variables de entrada (x_1 y x_2) se obtiene un 2. Cómo sólo en ese caso se desea que la salida, x_3 tome el valor 1 la restricción siguiente *casi* modeliza el comportamiento de la funcion AND:

$$x_1 + x_2 \geq 2x_3$$

porque en el caso $x_1 = x_2 = 1$, x_3 podría tomar uno cualquiera de los valores 0 ó 1. Para preservar la satisfacción de los casos resueltos por la restricción anterior y, además, forzar a x_3 a tomar el valor 1 cuando las dos entradas están activas se necesita la restricción adicional:

$$x_1 + x_2 \leq 2x_3 + 1$$

Ambas restricciones pueden reescribirse como sigue:

$$\begin{array}{rcrcrcl} x_1 & + & x_2 & - & 2x_3 & \geq & 0 \\ x_1 & + & x_2 & - & 2x_3 & \leq & 1 \\ & & & \mathbf{x} \in \{0,1\} & & & \end{array}$$

donde, como puede verse, las tres variables deben ser binarias.

Por último, el caso presentado en el primer apartado pedía, explícitamente, determinar el valor de las entradas para que la salida esté activada. Puesto que la activación se representa con el valor 1, y este es el máximo valor que puede tomar, la función objetivo debe ser:

$$\text{máx}\, z = x_3$$

resultando por fin la tarea de Programación Lineal que se muestra a continuación:

$$\begin{array}{rcrcrcl} & & & \text{máx}\, z = x_3 & & & \\ x_1 & + & x_2 & - & 2x_3 & \geq & 0 \\ x_1 & + & x_2 & - & 2x_3 & \leq & 1 \\ & & & \mathbf{x} \in \{0,1\} & & & \end{array}$$

2. Como en el apartado anterior, las variables serán las que representen el estado de cada una de las entradas —x_1, x_2— y las salidas, x_3, y que serán también binarias. La región factible que representa el comportamiento de una puerta lógica OR se muestra en la Figura 2. Como puede verse, x_3 toma el valor 1 si x_1 o x_2 valen 1, y 0 en caso contrario.

   En este caso, la puerta lógica OR se activa ($x_3 = 1$) si si una cualquiera de sus entradas están activas, esto es, si $x_1 = 1$ o $x_2 = 1$. Esto es, el valor de la salida debe ser siempre mayor o igual que cualquiera de sus entradas:

$$\begin{array}{rcl} x_1 & \leq & x_3 \\ x_2 & \leq & x_3 \end{array}$$

   ahora bien, esto no es suficiente puesto que estas restricciones permiten que x_3 tome el valor 1 siempre, independiente del valor de las entradas x_1 y x_2. Para evitar que la salida tome el valor 1 si las dos entradas están desactivadas debe añadirse:

$$x_1 + x_2 \geq x_3$$

   Todas estas restricciones deben reescribirse de la manera:

$$\begin{array}{rcrcrcl} x_1 & & & - & x_3 & \leq & 0 \\ & & x_2 & - & x_3 & \leq & 0 \\ x_1 & + & x_2 & - & x_3 & \geq & 0 \end{array}$$

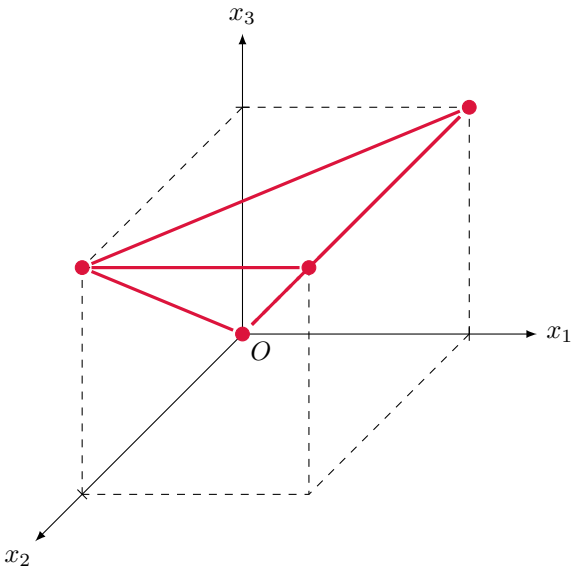


Figura 2: Región factible que simula el comportamiento de una puerta lógica OR

Nótese el valor complementario de las restricciones: las dos primeras sirven para activar la salida si cualquier entrada se ha activado; la tercera sirve para evitar que la salida se active si ninguna entrada lo está.

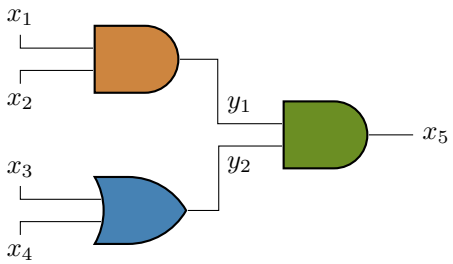
Por último, el problema pedía explícitamente determinar el valor de las entradas para que la salida esté desactivada, $x_3 = 0$. Puesto que 0 es el menor valor que puede tomar la variable x_3, la función objetivo consiste en minimizar x_3, resultando la tarea de Programación Lineal que se muestra a continuación:

$$\begin{array}{rcrcrcl} & & \min z = x_3 & & & & \\ x_1 & & & - & x_3 & \leq & 0 \\ & & x_2 & - & x_3 & \leq & 0 \\ x_1 & + & x_2 & - & x_3 & \geq & 0 \\ & & \mathbf{x} \in \{0,1\} & & & & \end{array}$$

3. Los dos apartados anteriores sirven para modelizar cada puerta independiente. La combinación de las restricciones de cada caso sirve ahora para modelizar circuitos como el mostrado en el tercer apartado. Para ello, es preciso individualizar las entradas y salida de cada puerta. Por lo tanto, se introducen dos variables adicionales:

   y_1 ∼ Salida de la primera puerta AND
   y_2 ∼ Salida de la primera puerta OR

   que son, a su vez, las entradas de la última puerta AND y que, como las demás, también serán binarias. La siguiente figura muestra esquemáticamente el significado de las dos nuevas variables:



Por lo tanto, la siguiente tarea de Programación Lineal combina las modelizaciones de las dos tareas anteriores después de hacer las sustituciones de variables que corresponden en cada caso:

$$\begin{array}{rcrcrcl} x_1 & + & x_2 & - & 2y_1 & \geq & 0 \\ x_1 & + & x_2 & - & 2y_1 & \leq & 1 \\ & & & & & & \\ x_3 & & & - & y_2 & \leq & 0 \\ & & x_4 & - & y_2 & \leq & 0 \\ x_3 & + & x_4 & - & y_2 & \geq & 0 \\ & & & & & & \\ y_1 & + & y_2 & - & 2x_5 & \geq & 0 \\ y_1 & + & y_2 & - & 2x_5 & \leq & 1 \end{array}$$

donde el color de cada restricción se corresponde con el de las puertas mostradas en la figura anterior.

Ahora bien, como se desea determinar el número mínimo de entradas que tienen que activarse para que la salida x_5 se active es preciso:

- Como todas las variables son binarias, la minimización de la suma de todas las variables x_i de entrada garantiza que sólo el número mínimo de ellas se activará, resultando la función objetivo: $\min z = \sum_{i=1}^{4} x_i$
- Por otra parte, como se desea que la salida x_5 tome el valor 1, aún es preciso añadir otra restricción adicional para forzar ese estado: $x_5 = 1$

Que da lugar entonces a la tarea de Programación Lineal que se muestra a continuación:

$$\min z = \sum_{i=1}^{4} x_i$$

$$\begin{array}{rcrcrcrcrcl} x_1 & + & x_2 & & & & & - & 2y_1 & & \geq & 0 \\ x_1 & + & x_2 & & & & & - & 2y_1 & & \leq & 1 \\ & & & & & & & & & & & \\ & & x_3 & & & & & & & - \; y_2 & \leq & 0 \\ & & & & x_4 & & & & & - \; y_2 & \leq & 0 \\ & & x_3 & + & x_4 & & & & & - \; y_2 & \geq & 0 \\ & & & & & & & & & & & \\ & & & & & - & 2x_5 & + & y_1 & + \; y_2 & \geq & 0 \\ & & & & & - & 2x_5 & + & y_1 & + \; y_2 & \leq & 1 \\ & & & & & & & & & & & \\ & & & & & & x_5 & & & & = & 1 \end{array}$$

$$\mathbf{x}, \mathbf{y} \in \{0, 1\}$$

Problema 2

1. Un problema de satisfacción de restricciones se define como una terna $\langle X, D, C \rangle$ donde $X = \{x_i\}_{i=1}^{n}$ es el conjunto de variables; $D = \{D_i\}_{i=1}^{n}$ representa los dominios de cada variable respectivamente y $C = \{C_i\}_{i=1}^{m}$ es el conjunto de restricciones del problema.

   A continuación se seguirá el orden convencional para la modelización basada en satisfacción de restricciones:

   a) Definición de las variables, X

   b) Cálculo del dominio de todas las variables, D

   c) Estudio de las restricciones entre las variables, C

   Variables El enunciado se refiere al estado de varios productos químicos, y sus combinaciones:

   $x_i \quad \sim \quad$ Estado del producto químico $i = \{\text{A}, \text{B}, \text{C}, \text{D}, \text{E}\}$

Dominios Inicialmente, los productos D y E no existen, mientras que los tres primeros sí están disponibles. Además, si un producto químico está disponible, entonces puede o bien estar en una probeta o vaso listo para ser mezclado con otro, o no. Por lo tanto, el estado de cualquier producto químico puede describirse con uno de los siguientes valores:

`no-existe`	:	El producto químico no está disponible
`fuera-probeta`	:	El producto existe pero no está en una probeta listo para mezclarse con otro
`en-probeta`	:	El producto existe y está en una probeta listo para mezclarse con otro

Nótese que el primer valor, `no-existe`, es excluyente con los otros dos. En particular, los productos químicos A, B y C existen y, por lo tanto, sus dominios son:

$$D_{\mathtt{A}} = D_{\mathtt{B}} = D_{\mathtt{C}} = \{\texttt{fuera-probeta, en-probeta}\}$$

Sin embargo, los productos D y E aún no han sido creados. Por lo tanto, sus dominios son:

$$D_{\mathtt{D}} = D_{\mathtt{E}} = \{\texttt{no-existe}, \texttt{fuera-probeta}, \texttt{en-probeta}\}$$

Ahora bien, el último valor, `en-probeta`, puede suprimirse del dominio de E, puesto que en ningún caso se considera la posibilidad de mezclarlo con ningún otro producto. A continuación se muestran definitivamente los dominios de cada variable, junto con un icono que los representa y que se usará para simplificar la presentación de las restricciones a continuación:

$$\begin{aligned}
D_A &= \{\texttt{fuera-probeta}(\boxtimes), \texttt{en-probeta}(\sqcup)\} \\
D_B &= \{\texttt{fuera-probeta}(\boxtimes), \texttt{en-probeta}(\sqcup)\} \\
D_C &= \{\texttt{fuera-probeta}(\boxtimes), \texttt{en-probeta}(\sqcup)\} \\
D_D &= \{\texttt{no-existe}(\otimes), \texttt{fuera-probeta}(\boxtimes), \texttt{en-probeta}(\sqcup)\} \\
D_E &= \{\texttt{no-existe}(\otimes), \texttt{fuera-probeta}(\boxtimes)\}
\end{aligned}$$

Restricciones Cada restricción R_{ij} representa todos los valores (tomados de los dominios de las variables x_i y x_j respectivamente) que son simultáneamente legales.

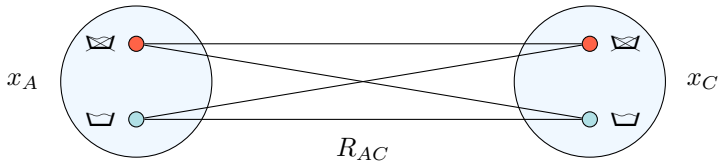
Producto	**A**	**B**	**C**	**D**	**E**
A		(⊠, ⊠), (⊠, ⊔), (⊔, ⊠)	(⊠, ⊠), (⊠, ⊔), (⊔, ⊠), (⊔, ⊔)	(⊠, ⊗), (⊠, ⊠), (⊠, ⊔), (⊔, ⊗), (⊔, ⊠)	(⊠, ⊗), (⊠, ⊠), (⊔, ⊗), (⊔, ⊠)
B			(⊠, ⊠), (⊠, ⊔), (⊔, ⊠)	(⊠, ⊗), (⊠, ⊠), (⊠, ⊔), (⊔, ⊗), (⊔, ⊠), (⊔, ⊔)	(⊠, ⊗), (⊠, ⊠), (⊔, ⊗), (⊔, ⊠)
C				(⊠, ⊗), (⊠, ⊠), (⊠, ⊔), (⊔, ⊗), (⊔, ⊠)	(⊠, ⊗), (⊠, ⊠), (⊔, ⊗), (⊔, ⊠)
D					(⊗, ⊗), (⊗, ⊠), (⊠, ⊗), (⊠, ⊠), (⊔, ⊗), (⊔, ⊠)

Puesto que ningún elemento tiene ninguna restricción con él mismo, los elementos de la diagonal principal han sido obviados. Además, para mejorar la legibilidad de la tabla, sólo se presentan las restricciones que hay sobre la diagonal principal, puesto que $R_{ij} = R_{ji}$.

Nótese que las restricciones representan valores que son simultaneamente legales, ¡aunque sean imposibles!. En particular, la tupla (⊗, ✉) $\in R_{DE}$ (mostrada en rojo en la tabla superior) es claramente imposible, puesto que si no existe D, no se podría haber generado E. Sin embargo, no es ilegal, puesto que ninguna restricción del problema la prohíbe expresamente.

2. La arco-consistencia entre dos variables x_i y x_j sirve para determinar si para cada valor $a_i \in D_i$ existe otro $a_j \in D_j$ que satisfaga la restricción que relaciona las variables i y j, R_{ij}. Si no fuera así, entonces a_i puede eliminarse del dominio de la primera variable, D_i.

   La siguiente figura representa la parte del *grafo de restricciones* que relaciona a las variables de interés en este apartado.

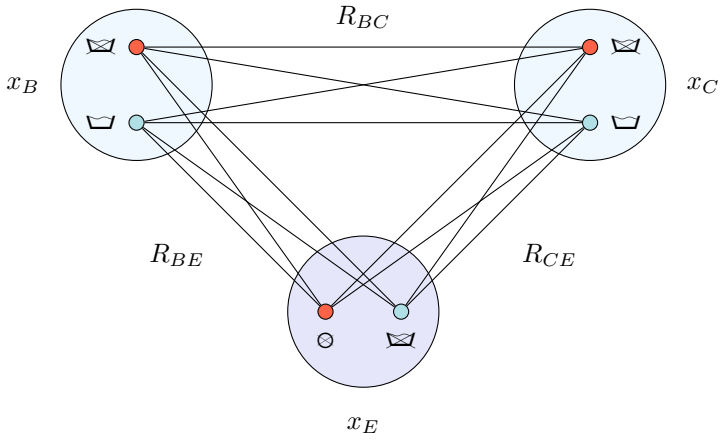


   Tal y como se vió en el apartado anterior, $R_{AC} = \{$(✉, ✉), (✉, ▭), (▭, ✉), (▭, ▭)$\}$ y, por otra parte, el dominio de las dos variables es el mismo e igual a $\{$✉, ▭$\}$. Por lo tanto, para cualquier valor de la primera variable, A, existe siempre un valor de la segunda, C, que aparece en la restricción que las une, R_{AC}. Además, lo contrario también es cierto, y para cada valor de la segunda variable, hay siempre un valor en el dominio de la primera, tal que ambos aparecen también en la misma restriccion, R_{AC}. Esto es así porque la restricción R_{AC} contiene el producto cartesiano del dominio de ambas variables.

   Por ello, las variables A y C son arco-consistentes (en cualquier dirección), y no ha sido preciso hacer ninguna modificación para forzar la arco-consistencia entre ellas.

3. Una variable x_i es camino-consistente con otra variable x_j, respecto de una tercera x_k si y sólo para cada par de valores legales entre las dos primeras variables, $(a_i, a_j) \in R_{ij}$, existe al menos un valor $a_k \in D_k$, tal que $(a_i, a_k) \in R_{ik}$, y $(a_j, a_k) \in R_{jk}$. Si no existe ningún valor a_k en el dominio de la variable x_k entonces la tupla (a_i, a_j) puede eliminarse de la restricción que relaciona las dos primeras variables.

   La siguiente figura representa la parte del *grafo de restricciones* que relaciona las tres variables de esta parte del ejercicio:



Las tuplas que hay en la restricción entre las variables x_B y x_C se muestran en la parte superior, R_{BC}. Como puede verse, para cualquier tupla, cualquier valor del dominio de x_E, está soportado por las restricciones R_{BE} y R_{CE}. Como ocurriera en el apartado anterior, esto es cierto, porque las tres restricciones son iguales al producto cartesiano de los dominios de las variables que relacionan.

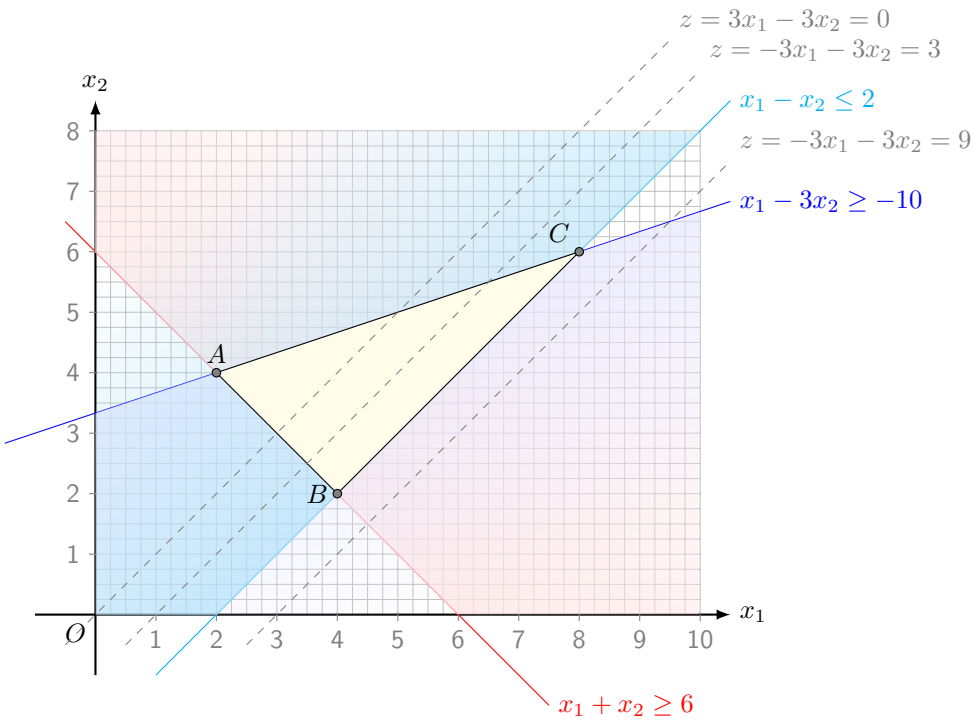
Por lo tanto, efectivamente x_B es camino-consistente con x_C respecto de x_E, y no ha sido preciso realizar ninguna modificación para forzarla.

Problema 3

Este es el mismo problema que el segundo del examen de junio de 2015, en el que se ha eliminado la última pregunta que se puso allí, para proponer aquí la parte (b).

1. Puesto que sólo hay dos variables de decisión, x_1 y x_2, las restricciones se pueden dibujar sobre un plano bidimensional.

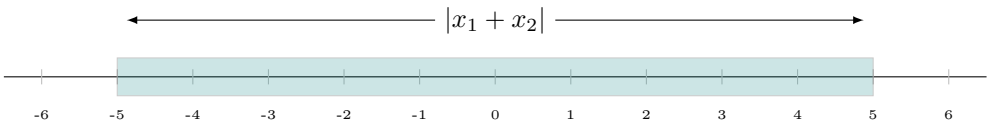
   La siguiente figura muestra gráficamente cada una de las restricciones. Como puede observarse, la región factible (mostrada en amarillo) consiste en la intersección de las regiones que satisfacen cada restricción (cada una dibujada de un color diferente), y está circunscrita por los puntos A, B y C.



   Las líneas discontinuas muestran las curvas de isobeneficio —denominadas así porque la función objetivo es de *maximización*. Se han dibujado con la intención de hacer aparente que los últimos puntos de la región factible que visita según va creciendo (puesto que se trata de un problema de maximización) son, precisamente, los de la recta $x_1 - x_2 = 2$ que delimita la región de la tercera restricción. Se trata, por lo tanto, de un problema con una cantidad infinita de soluciones óptimas: todos los puntos que hay en el segmento $\overline{BC}$.

   Como cualquiera de estos puntos verifican la recta $3x_1 - 3x_2 = 6$, el valor óptimo de la función objetivo es 6.

2. La siguiente figura muestra esquemática la restricción $|x_1 + x_2| \leq 5$:



que hace evidente que consiste en la satisfacción simultánea de las dos restricciones lineales siguientes:

$$\begin{array}{rcrcr} x_1 & + & x_2 & \geq & -5 \\ x_1 & + & x_2 & \leq & 5 \end{array}$$

Ahora bien, puesto que todas las variables de decisión son siempre estrictamente positivas, la primera restricción es redundante, y puede eliminarse, quedando entonces la tarea de Programación Lineal que se muestra a continuación:

$$\begin{array}{c} \text{máx}\, z = 3x_1 - 3x_2 \\ \begin{array}{rcrcr} x_1 & + & x_2 & \geq & 6 \\ x_1 & - & 3x_2 & \geq & -10 \\ x_1 & - & x_2 & \leq & 2 \\ x_1 & + & x_2 & \leq & 5 \end{array} \\ \mathbf{x} \geq \mathbf{0} \end{array}$$

3. Dada la nueva función objetivo, el problema de Programación Lineal que hay que considerar en este punto es el siguiente:

$$\begin{array}{c} \text{máx}\, z = 2x_1 - 6x_2 \\ \begin{array}{rcrcr} x_1 & + & x_2 & \geq & 6 \\ x_1 & - & 3x_2 & \geq & -10 \\ x_1 & - & x_2 & \leq & 2 \end{array} \\ \mathbf{x} \geq \mathbf{0} \end{array}$$

Un problema de programación lineal está en forma *estándar* si todas las restricciones son de igualdad, las variables de decisión son no negativas y, por último, el vector de constantes o recursos $\mathbf{b}$ no contiene términos negativos. Estará, además, en forma de maximización si la función objetivo maximiza y de minimización en otro caso. El problema, tal y como estaba enunciado, sólo verifica la segunda condición. Conviene aquí recordar:

- Una restricción de la forma $\leq$ está acotada superiormente. Puesto que ninguna variable de decisión puede tomar valores negativos, es preciso *sumar* una *variable de holgura* para forzar la igualdad.
- Análogamente, las restricciones de la forma $\geq$ están acotadas inferiormente de modo que, con variables de decisión que no pueden tomar valores negativos, es preciso *restar* una *variable de holgura* para forzar la igualdad.

Además, las variables de holgura que se añadan a las restricciones para forzar igualdades, se añadirán a la función objetivo z con un coeficiente nulo.

En primer lugar, se cambia el signo del recurso de la segunda restricción para hacer que sea positivo. Para ello, basta con multiplicar los dos términos de la segunda restricción por -1 (con lo que, además, se cambia el sentido de la desigualdad):

$$\begin{array}{c} \text{máx}\, z = 2x_1 - 6x_2 \\ \begin{array}{rcrcr} x_1 & + & x_2 & \geq & 6 \\ - \quad x_1 & + & 3x_2 & \leq & 10 \\ x_1 & - & x_2 & \leq & 2 \end{array} \\ \mathbf{x} \geq \mathbf{0} \end{array}$$

A continuación hay que añadir variables de holgura para convertir todas las desigualdades en igualdades. Por lo tanto, el problema de Programación Lineal queda, como sigue, en forma estándar de maximización:

$$\begin{array}{c} \text{máx } z = 2x_1 - 6x_2 \\ \begin{array}{rcrcrcrcrcl} & x_1 & + & x_2 & - & x_3 & & & & = & 6 \\ - & x_1 & + & 3x_2 & & & + & x_4 & & = & 10 \\ & x_1 & - & x_2 & & & & & + \; x_5 & = & 2 \end{array} \\ \mathbf{x} \geq \mathbf{0} \end{array}$$

4. El problema de Programación Lineal, tal y como se muestra en el apartado anterior, contiene dos columnas de la matriz identidad (de dimensión 3): $\mathbf{a_4}$ y $\mathbf{a_5}$. Para poder añadir la tercera columna que falta (y que consistirá primero de un 1 y luego ceros), se añade una *variable artificial* x_6:

$$\begin{array}{c} \text{máx } z = 2x_1 - 6x_2 - \infty x_6 \\ \begin{array}{rcrcrcrcrcl} & x_1 & + & x_2 & - & x_3 & & & + \; x_6 & = & 6 \\ - & x_1 & + & 3x_2 & & + & x_4 & & & = & 10 \\ & x_1 & - & x_2 & & & & + \; x_5 & & = & 2 \end{array} \\ \mathbf{x} \geq \mathbf{0} \end{array}$$

que se ha añadido también a la función objetivo con un coeficiente $-\infty$, puesto que su sentido es puramente técnico (se ha usado sólo para forzar la aparición de la matrix identidad como base factible inicial) y, por lo tanto, no debería aparecer en la solución final del SIMPLEX —salvo que el problema sea infactible, en cuyo caso, alguna variable artificial tomará un valor estrictamente positivo.

5. El algoritmo del SIMPLEX consiste en la aplicación iterativa de tres pasos: cálculo de las variables básicas, selección de la variable de entrada y selección de la variable de salida hasta que se detecte alguna de las siguientes condiciones:

   - El problema puede mejorar el valor de la función objetivo indefinidamente. Se dice entonces que el problema está *no acotado*. Este caso se detecta cuando todas las componentes y_i de la variable de decisión x_i elegida para entrar en la base son todos negativos o nulos.
   - El problema tiene soluciones infinitas. Este caso se detecta cuando los denominadores y_{ij} usados en la regla de salida para la variable que entra x_i son todos negativos o nulos.
   - El problema es irresoluble. Esto ocurre cuando en el segundo paso, todos los costes reducidos son positivos y el primer paso asignó un valor no negativo a alguna variable artificial.
   - Se alcanza una solución factible y puede demostrarse que no es posible mejorarla. Esta condición se detecta como en el segundo caso pero cuando las variables artificiales (si las hubiera) tienen valores nulos.

   Paso 0 Cálculo de una solución factible inicial

   a) Cálculo de las variables básicas
   La primera iteración se inicia con una base igual a la matriz identidad de dimensión 3, tal y como se calculó ya en el apartado anterior. Por lo tanto, son variables básicas en este paso $\{x_6, x_4, x_5\}$ (¡precisamente en este orden!):

$$B_0 = I_3 \qquad B_0^{-1} = I_3$$

$$x_0^* = B_0^{-1} b = b = \begin{pmatrix} 6 \\ 10 \\ 2 \end{pmatrix} \quad z_0^* = c_{B_0}^T x_0^* = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} \begin{pmatrix} 6 \\ 10 \\ 2 \end{pmatrix} = -6\infty$$

   Obviamente, $-6\infty = -\infty$. Sin embargo, ∞ se trata aquí simbólicamente como una variable con un valor arbitrariamente grande. Por lo tanto, es buena práctica preservar los coeficientes para poder hacer comparaciones con otras expresiones que también contengan el mismo término.

b) Selección de la variable de entrada
En las expresiones siguientes el cálculo de los vectores y_i se ha embebido en el cálculo de los *costes reducidos* directamente (aunque en una iteración con una base igual a la matriz identidad, $y_i = a_i$):

$$z_1 - c_1 = c_{B_0}^T B_0^{-1} a_1 - c_1 = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} - 2 = -\infty - 2$$

$$z_2 - c_2 = c_{B_0}^T B_0^{-1} a_2 - c_2 = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} 1 \\ 3 \\ -1 \end{pmatrix} + 6 = -\infty + 6$$

$$z_3 - c_3 = c_{B_0}^T B_0^{-1} a_3 - c_3 = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} - 0 = +\infty$$

Tal y como se indicaba anteriormente, ∞ se ha utilizado como un símbolo cualquiera. También es posible usar una constante M muy alta (y, en particular, un valor de M mayor que la suma de los valores absolutos de todos los coeficientes en la función objetivo). Usando ∞ como un símbolo cualquiera es posible saber qué valores son más grandes sustituyéndolo por valores arbitrariamente grandes.
En particular, sustituyendo ∞ por cualquier valor positivo arbitrariamente grande, x_1 será siempre más negativo que x_2 y ésta será, por tanto, la variable elegida para entrar en la siguiente base.

c) Selección de la variable de salida
La regla de salida establece que debe salir aquella variable con el menor cociente x_i/y_{ij} (con valores y_{ij} estrictamente positivos) donde x_i es la variable elegida en el paso anterior (x_1):

$$\text{mín} \left\{ \frac{6}{1}, \frac{10}{\cancel{-1}}, \frac{2}{1} \right\}$$

y sale la variable x_5 que es la que se corresponde con la tercera fracción (puesto que ocupa la tercera posición en la base).

Paso 1 Mejora de la solución actual (iteración #1)

a) Cálculo de las variables básicas
A continuación se mejora la calidad de la solución anterior. Las nuevas variables básicas son $\{x_1, x_4, x_6\}$. Nótese que ahora sí se han reordenado las variables con el único motivo de recordar su disposición más fácilmente.

$$B_1 = \begin{pmatrix} 1 & 0 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \qquad B_1^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix}$$

$$x_1^* = B_1^{-1} b = \begin{pmatrix} 2 \\ 12 \\ 4 \end{pmatrix} \quad z_1^* = c_{B_1}^T x_1^* = \begin{pmatrix} 2 & 0 & -\infty \end{pmatrix} \begin{pmatrix} 2 \\ 12 \\ 4 \end{pmatrix} = 4(1 - \infty)$$

Como aún hay una variable artificial en la base, su contribución al valor de la función objetivo es $-\infty$. Sin embargo, la función objetivo ha crecido (como debe ser, puesto que se trata de un problema de maximización) respecto de la iteración anterior de -6∞ a $4(1 - \infty)$.

b) Selección de la variable de entrada
Como antes, el cálculo de los vectores columna $y_i = B_1^{-1} a_i$ se ha embebido en el cálculo de los costes reducidos:

$$z_2 - c_2 = c_{B_1}^T B_1^{-1} a_2 - c_2 = \begin{pmatrix} 2 & 0 & -\infty \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ -1 \end{pmatrix} + 6 = -2\infty + 4$$

$$z_3 - c_3 = c_{B_1}^T B_1^{-1} a_3 - c_3 = \begin{pmatrix} 2 & 0 & -\infty \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} - 0 = +\infty$$

$$z_5 - c_5 = c_{B_1}^T B_1^{-1} a_5 - c_5 = \begin{pmatrix} 2 & 0 & -\infty \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - 0 \ = +\infty + 2$$

y la variable x_2 debe entrar en la base puesto que es la única que tiene un coste reducido estrictamente negativo.

c) Selección de la variable de salida
Como siempre, la variable de salida se calcula en atención al mínimo cociente x_1/y_{ij} (con valores y_{ij} estrictamente positivos) donde x_i es la variable elegida en el paso anterior para añadirse a la base (x_2) e y_{ij} son las componentes de su vector $\mathbf{y}$ también calculado en el paso anterior (aunque no se muestran explícitamente):

$$\text{mín} \left\{ \frac{2}{\cancel{-1}}, \frac{12}{2}, \frac{4}{2} \right\}$$

y sale la variable x_6, la variable artificial.

Paso 2 Mejora de la solución actual (iteración #2)

a) Cálculo de las variables básicas
Las nuevas variables básicas son $\{x_1, x_2, x_4\}$

$$B_2 = \begin{pmatrix} 1 & 1 & 0 \\ -1 & 3 & 1 \\ 1 & -1 & 0 \end{pmatrix} \qquad B_2^{-1} = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & -\frac{1}{2} \\ -1 & 1 & 2 \end{pmatrix}$$

$$x_2^* = B_2^{-1} b = \begin{pmatrix} 4 \\ 2 \\ 8 \end{pmatrix} \qquad z_2^* = c_{B_2}^T x_2^* = \begin{pmatrix} 2 & -6 & 0 \end{pmatrix} \begin{pmatrix} 4 \\ 2 \\ 8 \end{pmatrix} = -4$$

b) Selección de la variable de entrada
A continuación se muestra el cálculo de los costes reducidos de todas las variables no básicas, de las que se ha omitido únicamente, el de la variable artificial x_6 puesto que al hacer la diferencia con su coeficiente en la función objetivo resulta un valor positivo arbitrariamente grande:

$$z_3 - c_3 = c_{B_1}^T B_1^{-1} a_3 - c_3 = \begin{pmatrix} 2 & -6 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & -\frac{1}{2} \\ -1 & 1 & 2 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} - 0 = 2$$

$$z_5 - c_5 = c_{B_1}^T B_1^{-1} a_5 - c_5 = \begin{pmatrix} 2 & -6 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & -\frac{1}{2} \\ -1 & 1 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - 0 \ = 4$$

Como todos los costes reducidos son estrictamente positivos, añadir cualquiera de estas variables a la base decrementaría el valor de la función objetivo. En otras palabras, SIMPLEX ya ha encontrado la solución óptima.

Por lo tanto, la solución óptima, y el valor de la función objetivo en el punto óptimo son:

$$x^* = \begin{pmatrix} 4 \\ 2 \\ 0 \\ 8 \\ 0 \\ 0 \end{pmatrix} \quad z^* = -4$$

La solución óptima encontrada con SIMPLEX puede verificarse fácilmente en la figura del primer apartado. En particular, el punto x^* obtenido se corresponde, precisamente, con el punto C. Dibujando las curvas de isobeneficio consideradas en este apartado se puede ver fácilmente que ése es precisamente el último punto de la región factible que se visita en el sentido de crecimiento de la función objetivo.

6. La interpretación de un problema incluye varias consideraciones como son estudiar: si el problema es o no satisfacible, si la solución es única o hay varias soluciones o si está o no acotado. Además, debe estudiarse el uso de recursos: si sobra o no alguno y cual es su contribución al crecimiento de la función objetivo.

   Interpretación de la solución De la solución se puede advertir lo siguiente:

   - El problema es factible porque la solución no contiene valores positivos para ninguna variable artificial.
   - La solución es única porque los costes reducidos de la última iteración son todos estrictamente positivos. Eso significa que cualquier cambio en la base implicaría un decremento neto en el valor de la función objetivo.
   - El valor de la función objetivo está acotado porque siempre se pudo aplicar la regla de salida con denominadores estrictamente positivos.

   Interpretación de los recursos La interpretación de los recursos se hace, fundamentalmente, observando los valores de la variable de holgura en la solución óptima y la solución del problema dual:

   - Sobran hasta 8 unidades del segundo recurso, como lo atestigua el valor de la variable de holgura x_4 que, en el punto óptimo toma el valor 8. El resto de las variables de holgura valen 0 y eso significa que la solución óptima los explota por completo.
   - Por último, el crecimiento de la función objetivo por recurso se hace resolviendo el problema *dual*, donde cada variable x'_i representa el incremento de la función objetivo por unidad del recurso i-ésimo. Puesto que no se había pedido la resolución del problema dual, esta parte no era preciso cumplimentarla —sin embargo, el problema *dual* está resuelto en la solución del examen de Junio de 2015 donde puede consultarse.

Problema 4

1. El espacio de estados es una formalización que habilita la aplicación de algoritmos de búsqueda (informados o no) para la resolución de problemas. Consiste únicamente, en la definición de estados y operadores que sirven para transitar entre ellos. Relacionando, después, el conjunto de posibles estados con otro de vértices V y el de operadores (convenientemente instanciados) con otro de arcos E, resulta entonces de forma natural la definición de un grafo, el *grafo de búsqueda* que se recorrerá eficientemente con el uso de *árboles de búsqueda*. Este ejercicio propone ejercitar todos estos conceptos y, en el primer apartado, el del espacio de estados.

   Estados En este problema un estado se representa con información del estado del ascensor, los usuarios y todas las peticiones pendientes de ser atendidas. A continuación se presentan definiciones estructuradas para cada uno de estos conceptos:

Usuario Cada usuario debe identificarse de forma única con algún campo como su nombre o un código que le haga indistinguible de los demás, `usuario.id`. Además, tal y como advertía el enunciado, para cada uno se conoce la planta en la que se encuentra inicialmente (`usuario.from`), y aquella a la que desea llegar, `usuario.to`.

Obviamente, todos los campos indicados aquí son númericos —con la salvedad del primero, `usuario.id` que, tal vez, podría representarse con otros tipos diferentes.

Como eventualmente los usuarios irán entrando en el ascensor, puede parecer que debe crearse un campo para mantener esta información. En su lugar, se sugiere que sea el ascensor el que sepa quiénes están dentro de él.

Ascensor El ascensor tiene una capacidad máxima (`ascensor.C`) (que, en este caso, es igual a 10). Además, en cada estado podría encontrarse en una planta diferente, `ascensor.planta` y, podría llevar a un conjunto diferente de usuarios dentro, `ascensor.usuarios`.

Los dos primeros campos son necesariamente numéricos. Ahora bien, el último consiste simplemente en un *contenedor* de instancias de `usuario` descritas anteriormente.

Peticiones Las peticiones consisten, simplemente, en un *contendor* de instancias de `usuario`, cada una de las cuales ya mantiene información de la planta en la que se encuentre cada uno, y a la que desean acceder.

Sea `s` una instancia de estado representando con todas las estructura indicadas anteriormente que represente el estado inicial. El problema advertía explícitamente que en el estado inicial, el ascensor se encuentra en la primera planta, esto es, `s.ascensor.planta = 0`; asimismo, parece lógico asumir que no contendrá ningún pasajero dentro, de modo que `s.ascensor.usuarios=`$\varnothing$.

Sea `t` una instancia de estado que represente el estado final. En realidad, la única condición que se tiene que dar para asegurar que se ha alcanzado el estado final es `s.peticiones =` $\varnothing$ y, además, que no hay pasajeros en el ascensor (porque han sido todos despachados), `s.ascensor.usuarios=`$\varnothing$

Acciones En total pueden distinguirse tres acciones: una de ellas relacionadas con el movimiento del ascensor, y otras dos relacionadas con las operaciones con usuarios.

En cuanto al ascensor, basta con crear una única operacion que desplace el ascensor desde su planta actual hasta cualquier otra: `mover (planta_destino)`. Después de verificar que la planta de destino efectivamente existe, lo único que debe hacer es actualizar la planta en la que acabará el ascensor.

Ahora bien, también podría haberse modelado el movimiento del ascensor en pasos de un nivel cada uno, con dos operadores que no reciben parámetros: `subir` y `bajar`. Sus precondiciones y postcondiciones

En cuanto a los usuarios, estos pueden entrar o salir del ascensor. Para verificar que efectivamente pueden entrar basta con comprobar que el ascensor y el usuario están en la misma planta y, además, que no se viola la capacidad máxima. el único efecto de entrar en el ascensor es que son añadidos a la lista de usuarios dentro del ascensor. Análogamente, los usuarios pueden salir del ascensor si se encuentran dentro del ascensor y, además, este está en la misma planta que aquella a la que ellos quieran acceder, y tiene dos efectos: deben ser eliminados de la lista de usuarios en el ascensor y, además, de la lista de peticiones por atender, puesto que este cliente ya ha sido satisfecho.

La siguiente tabla muestra todas las acciones descritas anteriormente (donde `mover` es una alternativa al uso de las acciones de `subir` y `bajar`), caracterizando cada una con las *precondiciones* y *postcondiciones* necesarias.

Función	Precondiciones	Postcondiciones
`mover (planta_destino)`	`planta_destino` $\geq$ `0` $\wedge$ `planta_destino` $\leq$ `n`	`ascensor.planta` = `planta_destino`
`subir`	`ascensor.planta` $<$ `n`	`ascensor.planta`+ = `1`
`bajar`	`ascensor.planta` $>$ `0`	`ascensor.planta`− = `1`
`entrar (usuario)`	`ascensor.planta` = `usuario.from` $\wedge$ |`ascensor.usuarios`| $<$ `ascensor.C`	`insertar usuario en ascensor.usuarios`
`salir (usuario)`	`usuario` $\in$ `ascensor.usuarios` $\wedge$ `ascensor.planta` = `usuario.to`	`eliminar usuario de ascensor.usuarios y peticiones`

Además, es importante determinar el coste de cada una de estas acciones. Esto no se ha hecho explícitamente en la tabla anterior porque dependerá de la métrica que se use. De hecho, en otras secciones, el enunciado distingue entre dos:

Distancia Cuando pretenda minimizarse la distancia total recorrida por el ascensor, el coste de cada acción debe ser, entonces, la distancia que recorre.

En nuestra formalización anterior resulta obvio que `mover` tendrá entonces un coste que será igual al número de pisos que debe recorrer: `planta_destino` − `ascensor.planta`.

Alternativamente, las acciones de `subir` y `bajar` tienen todas un coste de una unidad, puesto que sólo desplazan el ascensor un piso.

Por otra parte, las acciones de carga y descarga de usuarios no puede tener coste, puesto que durante su ejecución el ascensor no se mueve.

TIempo En la minimización del tiempo deben entonces aplicarse los mismos costes que se explican en el cuarto apartado.

Esto es, el coste de mover el ascensor será 3 veces el número de pisos que se desplaza:

$$3 \times (\texttt{planta\_destino} - \texttt{ascensor.planta})$$

puesto que tarda 3 segundos en desplazarse cada nivel.

Sin embargo, las acciones de `subir` y `bajar` tienen un coste siempre igual a 3, puesto que sólo desplazan el ascensor un piso.

Por último, todas las operaciones de entrada y salida tendrán un coste de 30 segundos.

2. En este caso, el factor de ramificación máximo será igual al número de acciones que se han definido en el apartado anterior, convenientemente instanciadas.

   El número de personas que pueden subir en una planta es igual al número de personas que puede haber ahí, y éste número no puede ser mayor que el número de peticiones, |`peticiones`|.

   Análogamente, el número de personas que pueden bajar de un ascensor será, como máximo, el número de personas que hay dentro del ascensor, que no puede ser mayor que su capacidad —10, en el enunciado del problema.

   Por lo tanto, las acciones que se refieren a los usuarios pueden instanciarse como máximo de $10 +$ |`peticiones`| formas diferentes.

   Por último, si el movimiento del ascensor se modela con las acciones de `subir` y `bajar`, entonces deben añadirse otras dos acciones al cálculo del máximo factor de ramificación. Si, por el contrario, se modela con la acción de `mover`, entonces este operador podría instanciarse de $(n-1)$ formas diferentes, con n el número de plantas que tiene el edificio.

   En conclusión, habría hasta $12 +$ |`peticiones`| descendientes de cada nodo como máximo con el uso de `subir` y `bajar`, o hasta $10 + (n-1) +$ |`peticiones`| como factor de ramificación máximo si se emplea, en su lugar, el operador `mover`.

3. Tanto si el movimiento del ascensor se modeliza con las acciones de `subir` y `bajar`, como si se hace con la acción de `mover`, se trata de un problema de *costes variables*, puesto que las acciones de carga y descarga de personas tienen un coste nulo, distinto a los anteriores. Por ello, el algoritmo *del primero en amplitud* no encontraría ya soluciones óptimas, puesto que no existe una relación unívoca entre la profundidad a la que se encuentran las soluciones, y su coste.

   En su lugar, pueden usarse uno cualquiera de los algoritmos de *fuerza bruta* o *no informados* estudiados específicamente para el caso de costes variables:

   Ramificación y acotación en profundidad (DFBnB) Tiene un coste de memoria lineal pero puede re-expandir muchos nodos en caso de que el dominio presente muchas *transposiciones* (como ocurre con todos los algoritmos de *el primero en profundidad*).

   Dijkstra Consiste en un algoritmo de *el mejor primero* donde la función de evaluación, $f(n)$ es, simplemente, el coste del camino desde el estado inicial hasta n, $g(n)$. Tiene un coste de memoria exponencial pero garantiza que cada vez que expande un nodo habrá encontrado la solución óptima hasta él puesto que los nodos se expanden en orden creciente de su valor de $f(n)$ —o, equivalentemente, de $g(n)$.

   En este problema es particularmente fácil evitar las transposiciones con reglas muy sencillas. Además, es fácil garantizar que siempre se encontrará al menos una solución, aunque tenga un coste muy elevado. Por todo ello, cualquiera de los dos algoritmos anteriores servirían para encontrar soluciones óptimas.

4. No cambia absolutamente nada. Una vez más, se trata de acciones con costes diferentes (tanto si se modela el movimiento del ascensor con un grupo de acciones u otro) y, por lo tanto, las recomendaciones serían exactamente las mismas que las del apartado anterior.

5. Una heurística muy sencilla que puede obtenerse con la técnica de *relajación de restricciones*, consiste en relajar la capacidad del ascensor.

   Asumiendo ahora que la capacidad del ascensor es infinita, el *problema relajado* resultante puede resolverse óptimamente considerando la distancia que hay entre las plantas más baja y alta que haya que visitar. Como quiera que el ascensor debe ir a una planta, ya sea a recoger a un usuario, o a dejarle, la planta más baja a la que debe llegar el ascensor es:

   $$p^- = \min_{\forall \texttt{usuario} \in \texttt{peticiones}} \{\texttt{usuario.from}, \texttt{usuario.to}\}$$

   La misma consideración debe hacerse, naturalmente, para el cálculo de la planta más alta hasta la que hay que llevar el ascensor:

   $$p^+ = \max_{\forall \texttt{usuario} \in \texttt{peticiones}} \{\texttt{usuario.from}, \texttt{usuario.to}\}$$

   de donde resulta que la diferencia entre estas dos plantas es una estimación admisible (esto es, que no sobreestima el esfuerzo necesario):

   $$h_1 = p^+ - p^-$$

6. La selección de un algoritmo de búsqueda informada para este caso depende, como en el apartado anterior, del número de transposiciones y también, naturalmente, de la dificultad de los problemas:

   A* El algoritmo A* es admisible y garantiza, por lo tanto, que encontrará soluciones óptimas si la función heurística que lo guía también es admisible. Además, es un algoritmo rápido puesto que no reexpande nodos (y, con frecuencia, las ordenaciones de la lista abierta se pueden hacer en $O(1)$ con las estructuras de datos adecuadas si la función objetivo sólo toma valores enteros. Sin embargo, tiene un consumo de memoria exponencial.

IDA* El algoritmo IDA* reexpande nodos en caso de que haya transpociones pero no ordena nodos y, mucho más importante aún, tiene un consumo de memoria lineal en la profundidad de la solución (que en nuestro caso es siempre igual a N). Además, también es un algoritmo de búsqueda admisible.

Tal y como se indicó anteriormente parece particularmente fácil evitar las transposiciones en este dominio con reglas muy sencillas, de modo que cualquiera de estos algoritmos pueden usarse para encontrar soluciones óptimas al problema planteado.