Departamento de Informática
Universidad Carlos III de Madrid

Grado en Informática

Heurística y Optimización

9 de Enero de 2019

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

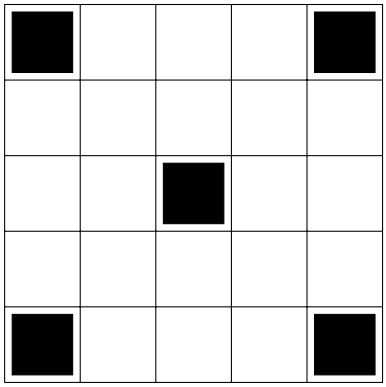
⑥ Si se sale del aula, no se podrá volver a entrar durante el examen

⑦ No se puede presentar el examen escrito a lápiz

Pregunta 1 (2 puntos)

Una compañía desea abrir k tiendas en una ciudad. Con el propósito de optimizar la cobertura en toda la ciudad, se pide disponer las k tiendas entre las n^2 posiciones candidatas representadas en una matrix de $n \times n$ posiciones tal que la distancia mínima entre cada par de tiendas se maximiza.

La siguiente figura muestra la solución óptima para $k = n = 5$:



Dados inicialmente valores para n y k, $(2 \leq k < n^2)$, se pide responder razonadamente las siguientes preguntas, explicando claramente en cada apartado todas las decisiones tomadas y su propósito, para resolver óptimamente este problema con una tarea de Programación Lineal:

(a) **(½ puntos)** En este problema es posible considerar diferentes distancias entre tiendas. Por ejemplo, se puede usar la distancia euclídea (d_E), o la distancia de Manhattan (d_M) definidas como sigue:

$$d_E(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$
$$d_M(P_1, P_2) = |x_1 - x_2| + |y_1 - y_2|$$

donde el punto P_i está representado por las coordenadas (x_i, y_i).

Dada una elección de la distancia a usar, ¿cómo se representarían las distancias entre dos puntos cualesquiera de una matriz de $n \times n$ en una tarea de Programación Lineal que sirva para resolver óptimamente este problema?

(b) **(½ puntos)** ¿Cuáles son las variables de decisión elegidas para resolver óptimamente este problema con una tarea de Programación Lineal?

(c) **(½ puntos)** ¿Cuáles son las restricciones de la tarea de Programación Lineal que sirven para resolver este problema óptimamente?

(d) **(½ puntos)** ¿Cuál es la función objetivo de la tarea de Programación Lineal que sirve para resolver este problema óptimamente?

Pregunta 2 (1½ puntos)

(a) **(½ puntos)** Dada una fórmula F en Forma Normal Conjuntiva $F = \bigwedge_{i=1}^{n} C_i$, considérese el caso de dos cláusula C_u y C_v que producen una, y sólo una cláusula, que es una tautología al aplicar la resolución respecto de un literal ℓ. ¿Es posible eliminar directamente esas cláusulas o deben usarse como cualquier otra durante la aplicación de un algoritmo de satisfabilidad lógica proposicional? Si o no y por qué.

Considera la fórmula F_1 en Forma Normal Conjuntiva $F_1 = \bigwedge_{i=1}^{5} C_i$ formada por las siguientes cláusulas:

$$\begin{array}{ll} C_1{:}(x_1 \vee x_3 \vee \bar{x}_4) & C_2{:}(\bar{x}_1 \vee x_2 \vee \bar{x}_4) \\ C_3{:}(x_2 \vee x_3) & C_4{:}(\bar{x}_2 \vee x_3 \vee x_4) \\ c_5{:}(\bar{x}_3 \vee x_4) & \end{array}$$

Se pide responder razonadamente las siguientes preguntas:

(b) **(½ puntos)** Aplicar Davis-Putnam (DP) para encontrar un modelo que satisfaga la fórmula F_1, en caso de que exista alguno.

Se exige aplicar DP usando las variables en orden ascendente de su subíndice

(c) **(½ puntos)** Aplicar Davis–Putnam–Logemann–Loveland (DPLL), y mostrar todos los modelos que satisfagan la fórmula F_1, en caso de que exista alguno.

Se exige aplicar DPLL usando las variables en orden ascendente de su subíndice

Pregunta 3 (3 puntos)

Considerése la siguiente tarea de Programación Lineal:

$$\text{mín}\, z = 7x_1 + 3x_2 - 8x_3$$

$$\begin{array}{rrrrrrrrr} & 5x_1 & + & 4x_2 & - & 2x_3 & \geq & & 3 \\ - & 4x_1 & + & 9x_2 & & & \geq & - & 3 \\ & 9x_1 & & & + & x_3 & \geq & - & 9 \end{array}$$

$$\mathbf{x} \geq \mathbf{0}$$

Se pide responder razonadamente las siguientes preguntas:

(a) **(½ puntos)** Expresar la tarea de Programación Lineal anterior en forma *estándar* de maximización.

(b) **(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.

(c) **(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

(d) **(½ puntos)** Calcular la contribución por unidad de recurso al crecimiento de la función objetivo de la tarea de Programación Lineal del enunciado.

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

Pregunta 4 (3½ puntos)

Un *JukeBox* consiste en una pila de n discos vinilo equipado con un brazo electrónico que puede extraer un disco de cualquier posición, entre la segunda y la última, para ponerlo en la primera. Dada una disposición inicial de los n discos, se está considerando la posibilidad de usar algoritmos de búsqueda para minimizar el número de movimientos del brazo de robot que disponen todos los discos en un orden específico.

Se pide responder razonadamente las siguientes preguntas:

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

(b) **(½ puntos)** ¿Cuál es el tamaño del espacio de estados?

(c) **(½ puntos)** ¿Cuál es el factor de ramificación mínimo desarrollado por cualquier algoritmo de búsqueda de fuerza bruta? ¿Y el máximo?

(d) **(½ puntos)** Si se desea calcular la solución óptima, ¿qué algoritmo de búsqueda de *fuerza bruta* sugerirías para su resolución? Por qué?

(e) **(1 punto)** Sugiere una función heurística $h(\cdot)$ que sea *admisible* e *informada* para el problema de minimizar el número de movimientos del brazo de robot necesarios para disponer los n discos en el orden especificado.

(f) **(½ puntos)** ¿Qué algoritmo de búsqueda *heurística* es el más indicado para resolver óptimamente el problema? ¿Por qué?

Soluciones del examen de Heurística y Optimización
Enero 2019

Problema 1

Este problema está tomado de una pregunta en *stack exchange*[1], y su propósito es el de determinar las diferentes partes de una tarea de Programación Lineal que sirven para resolver un problema de optimización: parámetros, variables, restricciones y, por último, función objetivo.

En las respuestas que se ofrecen a continuación, se discute también la implementación con MathProg con el único propósito de hacer más clara la exposición.

1. Para cualquier métrica de distancia usada en este problema, e independientemente de que consista en una expresión lineal o no, es posible representar la distancia entre dos pares cualesquiera de puntos de una rejilla de $n \times n$ con el uso de *parámetros*.

   Por lo tanto, basta con definir un parámetro ρ_{ij} que almacene la distancia entre el par de puntos i y j. Por supuesto, estos valores deben precalcularse y especificarse como variables constantes en la modelización del problema.

   Por ejemplo, sea `DISTANCE` el parámetro ρ_{ij} definido en MathProg para representar la distancia entre dos puntos cualesquiera:

   ```
   param DISTANCE {i in POINTS, j in POINTS};
   ```

   que, como puede verse, está definido sobre un conjunto de puntos, `POINTS`, que debe definirse también:

   ```
   set POINTS;
   ```

   Con ello, para la colección de puntos de una rejila de 3×3:

   ```
   set POINTS := P0 P1 P2 P3 P4 P5 P6 P7 P8;
   ```

   es posible especificar la distancia euclídea entre dos puntos cualesquiera como sigue:

   ```
   param DISTANCE : P0 P1 P2 P3 P4 P5 P6 P7 P8 :=
    P0  0.00 1.00 2.00 1.00 1.41 2.24 2.00 2.24 2.83
    P1  1.00 0.00 1.00 1.41 1.00 1.41 2.24 2.00 2.24
    P2  2.00 1.00 0.00 2.24 1.41 1.00 2.83 2.24 2.00
    P3  1.00 1.41 2.24 0.00 1.00 2.00 1.00 1.41 2.24
    P4  1.41 1.00 1.41 1.00 0.00 1.00 1.41 1.00 1.41
    P5  2.24 1.41 1.00 2.00 1.00 0.00 2.24 1.41 1.00
    P6  2.00 2.24 2.83 1.00 1.41 2.24 0.00 1.00 2.00
    P7  2.24 2.00 2.24 1.41 1.00 1.41 1.00 0.00 1.00
    P8  2.83 2.24 2.00 2.24 1.41 1.00 2.00 1.00 0.00;
   ```

   Puesto que `DISTANCE` contiene valores constantes, es posible usar una métrica arbitraria cualquiera.

2. Aparentemente es una buena idea definir una variable de decisión binaria x_i como sigue:

$$x_i \rightleftharpoons \begin{cases} 1 & \text{si la casilla } i \text{ contiene una tienda} \\ 0 & \text{en caso contrario} \end{cases}$$

[1] Ver `https://bit.ly/2s7vyw4`

Sin embargo, el problema pide expresamente maximizar la distancia mínima entre cada par de tiendas. Por lo tanto, es necesario conocer qué pares (i, j) de posiciones tienen tiendas asignadas para poder calcular la distancia entre ellas y, de ese modo, maximizar la distancia mínima entre todas ellas. Con el uso de una variable binaria como la sugerida sería necesario entonces, o bien añadir restricciones para asegurar que el cálculo de la distancia entre tiendas se hace únicamente entre pares de posiciones que tienen tiendas asignadas, o añadir variables binarias adicionales para determinar qué pares (i, j) tienen tiendas en ese par de posiciones.

En su lugar, se propone directamente el uso de una variable binaria x_{ij} definida como sigue:

$$x_{ij} \rightleftharpoons \begin{cases} 1 & \text{si y sólo si las casillas } i \text{ y } j \text{ tienen tiendas asignadas} \\ 0 & \text{en caso contrario} \end{cases}$$

Nótese que esta representación no tiene pérdida de completitud porque no es posible resolver este problema para $k = 1$. En otras palabras, es preciso que haya al menos un par de tiendas para que la métrica a optimizar tenga sentido.

De este modo, la propia definición de variables se puede utilizar para determinar los puntos cuyas distancias deben usarse en la función objetivo.

Estas variables de decisión se pueden representar en MathProg como sigue:

```
var enabled {i in POINTS, j in POINTS}, binary;
```

donde la variable x_{ij} se representa con `enabled [i, j]`.

3. En este problema deben considerarse tres tipos de restricciones, cuya redacción precisará de la definición de nuevas variables de decisión, tal y como se explica a continuación:

   Cálculo de la distancia mínima entre todos los pares de tiendas Lógicamente, la distancia mínima entre todos los pares de tiendas debe ser menor que la distancia entre cada par de puntos (i, j) que tengan una tienda asignada. Sea ρ_{ij} el parámetro definido anteriormente para almacenar la distancia entre un par de posiciones (i, j), y sea $d_<$ la distancia mínima entre todos los pares de tiendas. Es decir, $d_< = \text{mín}\{d_{ij}\}$, tal que $x_{ij} = 1$. La restricción:

   $$d_< \leq x_{ij}\rho_{ij}$$

   aparentemente resuelve el cálculo de la distancia mínima entre todos los pares de tiendas, pero no es cierto, porque obliga a la variable $d_<$ a tomar el valor 0 por cada par de posiciones (i, j) que no tengan una tienda asignada —puesto que para ellas $x_{ij} = 0$. Ahora bien, si se hace que la distancia entre un par de posiciones (i, j) para las que $x_{ij} = 0$ tome un valor arbitrariamente grande (puesto que $d_<$ debe ser *menor* que las distancias sólo entre tiendas, y no entre posiciones vacías), entonces $d_<$ se calcularía correctamente. Para ello, basta con añadir $(1 - x_{ij})M$, donde M es cualquier constante arbitrariamente grande:

   $$d_< \leq x_{ij}\rho_{ij} + (1 - x_{ij})M$$

   La variable $d_<$ se puede representar con una variable de decisión `min_distance` en MathProg:

   ```
   var min_distance;
   ```

   de modo que la restricción anterior se representa en MathProg como sigue:

   ```
   s.t. best_distance {i in POINTS, j in POINTS}:
      min_distance <= enabled[i, j] * DISTANCE[i, j] + (1 - enabled[i, j]) * 10000.0;
   ```

Consistencia de la variable de decisión A menos que se especifique explícitamente lo contrario, la resolución de la tarea de Programación Lineal podría dar valores 0 ó 1 a cualesquiera de las variables x_{ij}. Sin embargo, es obvio que si (i, j) son dos posiciones con tiendas asignadas, entonces (j, i) también lo son:

$$x_{ij} = x_{ji}$$

Esta restricción se representa en MathProg simplemente como sigue:

```
s.t. symmetry {i in POINTS, j in POINTS}:
   enabled [i, j] = enabled [j, i];
```

Selección de k tiendas Por último, es preciso asegurarse de que la solución de la tarea de Programación Lineal consistirá en exactamente k posiciones elegidas.

Con k posiciones elegidas, habrá exactamente $2\binom{k}{2} = k(k-1)$ variables x_{ij} que tomarán el valor 1 —donde el 2 que multiplica al número combinatorio $\binom{k}{2}$ resulta de la restricción de consistencia anterior. Por lo tanto:

$$\sum_{ij} x_{ij} = k(k-1);$$

aparentemente resuelve el problema. Sin embargo, no es así, y a menos que se añadan otras restricciones, es posible obtener soluciones factibles, pero incorrectas, definidas sobre un número de posiciones diferentes de k. Por ejemplo, en una rejilla de 3×3 como la definida en el primer apartado de este problema, sería factible obtener exactamente 6 variables binarias con el valor 1: $x_{08} = x_{80} = x_{26} = x_{62} = x_{56} = x_{65} = 1$ que, sin embargo, es incorrecta, puesto que está definida sobre 5 puntos (los puntos 0, 2, 5, 6 y 8), en vez de 3 cómo se pedía.

En su lugar, imagínenese una matriz $\mathcal{M}$ de $n^2 \times n^2$ posiciones donde la casilla $\mathcal{M}_{ij} = x_{ij}$. En una asignación correcta de k tiendas a k posiciones de la matriz $n \times n$ que representa a la ciudad, la suma de filas i (o posiciones de la matriz) que tienen al menos una columna j tal que $x_{ij} = 1$ debe ser exactamente igual a k.

Para demostrarlo, se muestra a continuación la matriz que resulta con la asignación factible (pero incorrecta) expuesta anteriormente:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} \mathbf{1} \\ 0 \\ \mathbf{1} \\ 0 \\ 0 \\ \mathbf{1} \\ \mathbf{1} \\ 0 \\ \mathbf{1} \end{matrix}$$

donde, como se ve, hay 5 filas, en vez de 3, con el valor 1. Por lo tanto, sea r_i, una variable de decisión binaria que toma el valor 1 sí y sólo si, la posición i tiene una tienda asignada, de modo que al menos para alguna posición j, $x_{ij} = 1$. De esta manera, la restricción:

$$\sum_{i} r_i = k$$

sí que garantiza que se escogerán, exactamente, k posiciones para instalar una tienda. Ahora sólo falta añadir una última restricción para que la variable r_i tome el valor correcto:

$$Mr_i \geq \sum_j x_{ij}$$

donde M es una constante arbitrariamente grande. Nótese, de hecho, la importancia de usar la constante M: puesto que r_i es una variable binaria, sólo puede tomar valores del conjunto $\{0, 1\}$. Ahora bien, una posición i podría tener que compararse con más de una posición j, y por lo tanto $\sum_j x_{ij}$ puede ser mayor que 1, lo que crearía una infactibilidad. Para evitarlo, basta con multiplicar r_i por la constante M.

Estas tres restricciones se modelan en MathProg de la siguiente manera:

```
s.t. sum_rows_lower {i in POINTS}:
   10000.0 * rows [i] >= sum {j in POINTS} enabled [i, j];

s.t. max_rows:
  sum {i in POINTS} rows[i] = TOTAL;

s.t. locations:
  sum {i in POINTS, j in POINTS} enabled[i, j] = TOTAL * (TOTAL-1);
```

donde `TOTAL` es el valor de k, que debe definirse como un parámetro:

```
param TOTAL;
```

y `rows [i]` es la variable binaria r_i que debe declararse como sigue:

```
var rows {i in POINTS}, binary;
```

4. La variable $d_<$ usada en el apartado anterior calcula la distancia mínima entre todos los pares de posiciones de la ciudad. Puesto que el enunciado solicitaba explícitamente maximizarla, la función objetivo es simplemente:

$$\text{máx}\, z = d_<$$

cuya modelización en MathProg es:

```
maximize DISTRIBUTION:
   min_distance;
```

La solución de este modelo con un fichero de datos que contenga las distancias euclídeas entre todos los pares de nodos de una rejilla de 5×5 da precisamente la solución mostrada en el enunciado.

Problema 2

1. Si la resolución de dos cláusulas C_u y C_v respecto de un literal ℓ produce una, y sólo una cláusula, que es una tautología, entonces la resolución no produce ningún conocimiento útil adicional. Sin embargo, eso no significa que puedan eliminarse directamente, puesto que entonces no se podría saber qué valores hay que asignar a las variables que contienen.

   Esto es particularmente cierto si las dos cláusulas involucradas C_u y C_v contienen algunas variables que no están presentes en ninguna otra cláusula. Considérese, por ejemplo, la siguiente fórmula en Forma Normal Conjuntiva, que consiste en las cláusulas:

$$\begin{array}{ll} C_1{:}(x_1 \vee x_2) & C_2{:}(\bar{x}_1 \vee \bar{x}_2) \\ C_3{:}(x_3 \vee x_4) & C_4{:}(\bar{x}_3 \vee x_4 \vee x_5) \end{array}$$

Obviamente, la resolución de C_1 y C_2 respecto de uno cualquiera de los literales x_1 o x_2 produce una tautología —por ejemplo, si se hace respecto de x_1 resulta la tautología $(x_2 \vee \bar{x}_2)$. Si se elimnan estas dos cláusulas, entonces la fórmula resultante F' consistiría en la conjunción de las cláusulas C_3 y C_4, que no contienen ni x_1, ni x_2. Pero no es cierto que un modelo que satisfaga la fórmula $F' = C_3 \wedge C_4$ también satisface la fórmula original F para cualquier asignación de los valores $\{\top, \bot\}$ a las variables x_1 y x_2. Por ejemplo, $\{x_1 = \top, x_2 = \top\}$ satisface C_1, pero no C_2 y, por lo tanto, no puede ser parte de un modelo que satisfaga la fórmula original.

2. Para aplicar Davis-Putnam, se escoge en cada iteración un literal y se aplica la resolución a la red de cláusulas actual respecto de ese literal. En cada paso, se guardan las variables usadas y las cláusulas involucradas de modo que si eventualmente resultara el conjunto vacío, $\varnothing$, se usa esa información para generar un modelo que valide la expresión inicial. Si, en otro caso, se obtiene la cláusula vacía, $\{\varnothing\}$, entonces se habrá probado que la fórmula original es insatisfacible, concluyendo así la aplicación del algoritmo —puesto que ahora se sabe que no hay ningún modelo que calcular.

   A continuación se muestran todos los pasos del algoritmo DP. En cada paso se muestran la red de cláusulas G_i, la variable empleada x_j (que será, como se solicita en el enunciado, la siguiente en orden ascendiente de su subíndice), y las cláusulas involucradas, que se calculan como la diferencia entre la red de cláusulas de cada paso y el resultado de aplicar resolución: $G_i \backslash \mathrm{Res}(G_i, x_j)$.

Paso 0 $G_0 = \bigwedge_{i=1}^{4} C_i$

La primera variable a utilizar es x_1. Aplicando el método de resolución respecto de élla, se observa que el literal x_1 aparece afirmado en C_1, y negado en C_2. Después de calcular la disyunción de los literales que acompañan al literal afirmado y negado en el par indicado, resulta la cláusula $C_6 : (x_2 \vee x_3 \vee \bar{x}_4)$ que, como no es una tautología se añade al *diccionario* de cláusulas:

$$\mathrm{Res}(G_0, x_1) = \{C_3, C_4, C_5, C_6 : (x_2 \vee x_3 \vee \bar{x}_4)\}$$

donde, como se ve, las cláusulas C_3, C_4 y C_5 pasan inalteradas puesto que no contienen el literal x_1.

Las cláusulas involucradas en este paso se calculan con la expresión:

$$G_0 \backslash \mathrm{Res}(G_0, x_1) = G_0 \backslash \{C_3, C_4, C_5, C_6\} = \{C_1, C_2\}$$

Paso 1 $G_1 = \bigwedge_{i=3}^{6} C_i$

En el segundo paso de aplicación del algoritmo Davis-Putnam debe utilizarse x_2, que aparece afirmado en C_3 y C_6, y negado en C_4. Por lo tanto, es preciso considerar la resolución de los pares de cláusulas $\{C_3, C_4\}$ y $\{C_6, C_4\}$ respecto del literal x_2.

El primer par produce la cláusula $C_7 : (x_3 \vee x_4)$ que se añade al *diccionario* de cláusulas puesto que no es una tautología. Sin embargo, la resolución de C_6 y C_4 produce $(x_3 \vee x_4 \vee \bar{x}_4)$ que, al ser una tautología es ignorada.

Por lo tanto:

$$\mathrm{Res}(G_1, x_2) = \{C_5, C_7 : (x_3 \vee x_4)\}$$

donde C_5 se incluye en el resultado de la resolución calculada puesto que no contiene al literal x_2.

En este paso, las cláusulas involucradas son, por lo tanto:

$$G_1 \backslash \mathrm{Res}(G_1, x_2) = G_1 \backslash \{C_5, C_7\} = \{C_3, C_4, C_6\}$$

Paso	Variable	Cláusulas
3	x_4	$C_8 : (x_4)$
2	x_3	$C_5 : (\bar{x}_3 \vee x_4), C_7 : (x_3 \vee x_4)$
1	x_2	$C_3 : (x_2 \vee x_3), C_4 : (\bar{x}_2 \vee x_3 \vee x_4), C_6 : (x_2 \vee x_3 \vee \bar{x}_4)$
0	x_1	$C_1 : (x_1 \vee x_3 \vee \bar{x}_4), C_2 : (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

Cuadro 1: Fase II del algoritmo de Davis-Putnam

Paso 2 $G_2 = \{C_5, C_7\}$

El enunciado pedía explícitamente considerar ahora el literal x_3, que aparece afirmado en C_7, y negado en C_5, resultando:

$$\text{Res}(G_2, x_3) = \{C_8 : (x_4)\}$$

Nótese que, en este caso, no se han añadido cláusulas adicionales que no contengan al literal empleado en este paso, puesto que todas las cláusulas de la red de restricciones actuales contienen al literal considerado, x_3. Con ello, las cláusulas involucradas se calculan como sigue:

$$G_2 \backslash \text{Res}(G_2, x_3) = G_2 \backslash \{C_8\} = \{C_5, C_7\}$$

Paso 3 $G_3 = \{C_8\}$

Obviamente, la red de restricciones de este paso es trivialmente satisfacible, pero aún así es preciso proceder en el mismo orden que en los pasos anteriores. En este caso, la variable x_4 es un literal puro en la red de restricciones G_3 y, por lo tanto, simplemente desaparece:

$$\text{Res}(G_3, x_4) = \varnothing$$

con lo que se prueba que la fórmula original es satisfacible, pero antes de pasar al cálculo del modelo, se calculan también en este último paso las cláusulas involucradas:

$$G_3 \backslash \text{Res}(G_3, x_4) = \{C_8\}$$

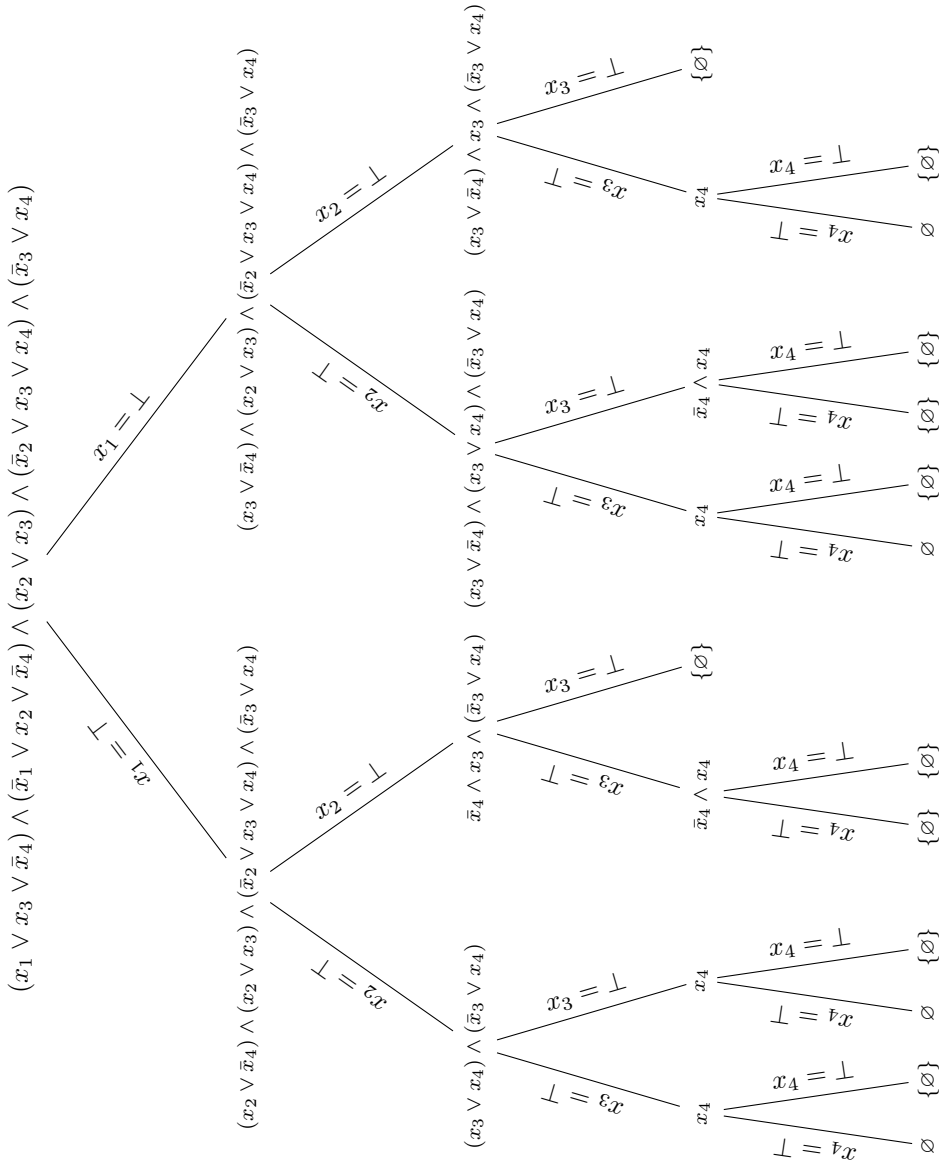
Puesto que la fórmula es lógicamente satisfacible, debe procederse ahora a la segunda fase del algoritmo de Davis-Putnam, que considera todas las variables empleadas y las cláusulas involucradas en cada paso en orden inverso al de su aplicación. La tabla 1 muestra esta información.

En la segunda fase del algoritmo de Davis-Putnam, un modelo que valide una expresión satisfacible de la lógica proposicional puede encontrarse componiendo los modelos que satisfacen las cláusulas involucradas en cada paso dando valores a las variables empleadas y, sólo si fuera preciso, a otras. Por supuesto, en cada paso, deben tenerse en cuenta las asignaciones realizadas en los pasos anteriores.

Empezando con el paso 3, basta con hacer $x_4 = \top$ para satisfacer C_8. En el paso 2, se observa que C_7 se valida automáticamente con la asignación del paso anterior, de modo que haciendo ahora $x_3 = \bot$, se satisface C_5. Como $x_4 = \top$, la cláusula C_4 es cierta ya, y basta ahora con hacer $x_2 = \top$ para satisfacer las cláusulas C_3 y C_6 involucradas en el paso 1. Por último, la cláusula C_2 ya está validada puesto que $x_2 = \top$, de modo que sólo falta hacer $x_1 = \top$ para validar la última cláusula involucrada en el paso 0, C_1.

Por lo tanto, el modelo resultante es:

$$M_1 = \{x_1 = \top, x_2 = \top, x_3 = \bot, x_4 = \top\}$$


$(x_1 \vee x_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge (\bar{x}_3 \vee x_4)$
$x_1 = \top$
$\top = x_1$
$(x_3 \vee \bar{x}_4) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge (\bar{x}_3 \vee x_4)$
$(x_2 \vee \bar{x}_4) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge (\bar{x}_3 \vee x_4)$
$x_2 = \top$
$\top = x_2$
$x_2 = \top$
$\top = x_2$
$(x_3 \vee \bar{x}_4) \wedge x_3 \wedge (\bar{x}_3 \vee x_4)$
$(x_3 \vee \bar{x}_4) \wedge (x_3 \vee x_4) \wedge (\bar{x}_3 \vee x_4)$
$\bar{x}_4 \wedge x_3 \wedge (\bar{x}_3 \vee x_4)$
$(x_3 \vee x_4) \wedge (\bar{x}_3 \vee x_4)$
$x_3 = \top$
$\top = \varepsilon_x$
$x_3 = \top$
$\top = \varepsilon_x$
$x_3 = \top$
$\top = \varepsilon_x$
$x_3 = \top$
$\top = \varepsilon_x$
$\{\varnothing\}$
x_4
$\bar{x}_4 \wedge x_4$
x_4
x_4
$x_4 = \top$
$\top = x_4$
$x_4 = \top$
$\top = x_4$
$x_4 = \top$
$\top = x_4$
$x_4 = \top$
$\top = x_4$
$\{\varnothing\}$
$\varnothing$
$\{\varnothing\}$
$\{\varnothing\}$
$\{\varnothing\}$
$\{\varnothing\}$
$\{\varnothing\}$
$\varnothing$

3. El siguiente árbol muestra la aplicación del algoritmo Davis-Putnam-Logemann-Loveland (DPLL) a la fórmula F_1 en orden creciente de los subíndices de las variables, tal y como se solicitaba en el enunciado:

   Como puede verse, hay en total hasta cuatro modelos diferentes, uno de los cuales M_2, es el mismo obtenido en el apartado anterior:

$$M_1{=}\{x_1 = \top, x_2 = \top, x_3 = \top, x_4 = \top\}$$
$$M_2{=}\{x_1 = \top, x_2 = \top, x_3 = \bot, x_4 = \top\}$$
$$M_3{=}\{x_1 = \bot, x_2 = \top, x_3 = \top, x_4 = \top\}$$
$$M_4{=}\{x_1 = \bot, x_2 = \bot, x_3 = \top, x_4 = \top\}$$

Problema 3

1. 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 y tercera restricciones para hacer que sean positivos. Para ello, basta con multiplicar los dos términos de cada restricción por -1 (con lo que, además, se cambia el sentido de la desigualdad):

$$\text{mín } z = 7x_1 + 3x_2 - 8x_3$$
$$\begin{array}{rcrcrcl} 5x_1 & + & 4x_2 & - & 2x_3 & \geq & 3 \\ 4x_1 & - & 9x_2 & & & \leq & 3 \\ -\ 9x_1 & & & - & x_3 & \leq & 9 \end{array}$$
$$\mathbf{x} \geq \mathbf{0}$$

   A continuación se añaden variables de holgura para convertir todas las desigualdades en igualdades tal y como se explicaba anteriormente. Por lo tanto, el problema de Programación Lineal queda, como sigue:

$$\text{mín } z = 7x_1 + 3x_2 - 8x_3$$
$$\begin{array}{rcrcrcrcrcrcl} 5x_1 & + & 4x_2 & - & 2x_3 & - & x_4 & & & & & = & 3 \\ 4x_1 & - & 9x_2 & & & & & + & x_5 & & & = & 3 \\ -\ 9x_1 & & & - & x_3 & & & & & + & x_6 & = & 9 \end{array}$$
$$\mathbf{x} \geq \mathbf{0}$$

   Por último, la función objetivo debe ponerse en la forma de maximización. Para ello, basta con multiplicarla por -1:

$$\text{máx}\, z = -7x_1 - 3x_2 + 8x_3$$

$$\begin{array}{rcrcrcrcrcrcl} 5x_1 & + & 4x_2 & - & 2x_3 & - & x_4 & & & & & = & 3 \\ 4x_1 & - & 9x_2 & & & & & + & x_5 & & & = & 3 \\ -\ 9x_1 & & & - & x_3 & & & & & + & x_6 & = & 9 \end{array}$$

$$\mathbf{x} \geq \mathbf{0}$$

2. El problema de Programación Lineal, tal y como se muestra en el apartado anterior, contiene dos columnas de la matriz identidad (de dimensión 3): $\mathbf{a_5}$ y $\mathbf{a_6}$. Para poder añadir la columna que falta (y que consistirá en un uno, y luego dos ceros), se añade una *variable artificial* x_7:

$$\text{máx}\, z = -7x_1 - 3x_2 + 8x_3 - \infty x_7$$

$$\begin{array}{rcrcrcrcrcrcl} 5x_1 & + & 4x_2 & - & 2x_3 & - & x_4 & & & & + & x_7 & = & 3 \\ 4x_1 & - & 9x_2 & & & & & + & x_5 & & & & = & 3 \\ -\ 9x_1 & & & - & x_3 & & & & & + & x_6 & & = & 9 \end{array}$$

$$\mathbf{x} \geq \mathbf{0}$$

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, la variable artificial tomará un valor estrictamente positivo.

3. 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 es infactible. 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.
   - El problema tiene soluciones infinitas. En este caso todos los costes reducidos calculados en el segundo paso son positivos y hay, al menos, uno con coste nulo. La arista que une el punto calculado en la iteración actual con aquél que se alcanzaría si entra una variable con coste reducido nulo (cuando no hay costes reducidos negativos) son todas soluciones óptimas de la tarea de Programación Lineal.

   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 $\{x_7, x_5, x_6\}$, precisamente en este orden puesto que son precisamente las columnas $\mathbf{a_7}$, $\mathbf{a_5}$ y $\mathbf{a_6}$ las que recrean la matriz identidad de rango 3, I_3:

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

$$x_0^* = B_0^{-1} b = b = \begin{pmatrix} 3 \\ 3 \\ 9 \end{pmatrix} \quad z_0^* = c_{B_0}^T x_0^* = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} \begin{pmatrix} 3 \\ 3 \\ 9 \end{pmatrix} = -3\infty$$

Obviamente, $-3\infty = -\infty$. 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, $y_i = a_i$):

$$z_1 - c_1 = c_{B_0}^T B_0^{-1} a_1 - c_1 = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} 5 \\ 4 \\ -9 \end{pmatrix} + 7 = -5\infty + 7$$

$$z_2 - c_2 = c_{B_0}^T B_0^{-1} a_2 - c_2 = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} 4 \\ -9 \\ 0 \end{pmatrix} + 3 = -4\infty + 3$$

$$z_3 - c_3 = c_{B_0}^T B_0^{-1} a_3 - c_3 = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} -2 \\ 0 \\ -1 \end{pmatrix} - 8 = 2\infty - 8$$

$$z_4 - c_4 = c_{B_0}^T B_0^{-1} a_4 - c_4 = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} + 0 = +\infty$$

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, el coste reducido de x_1 será siempre más negativo que el del resto de las variables no básicas y, por tanto, la variable elegida para entrar en la siguiente base será percisamente x_1.

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):

$$\text{mín}\left\{\frac{3}{5}, \frac{3}{4}, \frac{9}{\cancel{-9}}\right\}$$

y sale la variable x_7 que es la que se corresponde con la primera fracción (puesto que ocupa la primera posición en la base), precisamente la variable artificial.

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 $\{x_1, x_5, x_6\}$:

$$B_1 = \begin{pmatrix} 5 & 0 & 0 \\ 4 & 1 & 0 \\ -9 & 0 & 1 \end{pmatrix} \qquad B_1^{-1} = \begin{pmatrix} \frac{1}{5} & 0 & 0 \\ -\frac{4}{5} & 1 & 0 \\ \frac{9}{5} & 0 & 1 \end{pmatrix}$$

$$x_1^* = B_1^{-1} b = \begin{pmatrix} \frac{3}{5} \\ \frac{3}{5} \\ \frac{72}{5} \end{pmatrix} \quad z_1^* = c_{B_1}^T x_1^* = \begin{pmatrix} -7 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{3}{5} \\ \frac{3}{5} \\ \frac{72}{5} \end{pmatrix} = -\frac{21}{5}$$

Como puede verse, la función objetivo ha crecido (como debe ser, puesto que se trata de un problema de maximización) respecto de la iteración anterior de -3∞ a $-\frac{21}{5}$.

b) Selección de la variable de entrada
Primero, se calculan los vectores columna $y_i = B_1^{-1}a_i$:

$$y_2{=}B_1^{-1}a_2{=}\begin{pmatrix} \frac{4}{5} \\ -\frac{61}{5} \\ \frac{36}{5} \end{pmatrix} \quad y_3{=}B_1^{-1}a_3{=}\begin{pmatrix} -\frac{2}{5} \\ \frac{8}{5} \\ -\frac{23}{5} \end{pmatrix} \quad y_4{=}B_1^{-1}a_4{=}\begin{pmatrix} -\frac{1}{5} \\ \frac{4}{5} \\ -\frac{9}{5} \end{pmatrix}$$

Con ellos, ya es posible calcular los costes reducidos de las variables no básicas candidatas a entrar en la base en la siguiente iteración::

$$\begin{aligned} z_2 - c_2{=}c_{B_1}^T B_1^{-1}a_2 - c_2{=}\begin{pmatrix} -7 & 0 & 0 \end{pmatrix} y_2 + 3{=}-\tfrac{13}{5} \\ z_3 - c_3{=}c_{B_1}^T B_1^{-1}a_3 - c_3{=}\begin{pmatrix} -7 & 0 & 0 \end{pmatrix} y_3 - 8{=}-\tfrac{26}{5} \\ z_4 - c_4{=}c_{B_1}^T B_1^{-1}a_4 - c_4{=}\begin{pmatrix} -7 & 0 & 0 \end{pmatrix} y_4 + 0{=}\tfrac{7}{5} \end{aligned}$$

y la variable x_3 debe entrar en la base puesto que es la que tiene el coste reducido negativo de mayor mangitud. Nótese que en este paso no se ha calculado el coste reducido de la variable artificial, x_7 puesto que siempre sumará $+\infty$ y, por lo tanto, es claramente imposible que vuelva a la base.

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_3) e y_{ij} son las componentes de su vector $\mathbf{y}$

$$\text{mín}\left\{ \frac{\frac{3}{5}}{\cancel{\frac{2}{5}}}, \frac{\frac{3}{5}}{\frac{8}{5}}, \frac{\cancel{\frac{72}{5}}}{\cancel{\frac{23}{5}}} \right\}$$

y sale la variable x_5, que es la que ocupa la segunda posición en la base actual.

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

a) Cálculo de las variables básicas
Las nuevas variables básicas son $\{x_1, x_3, x_6\}$

$$B_2 = \begin{pmatrix} 5 & -2 & 0 \\ 4 & 0 & 0 \\ -9 & -1 & 1 \end{pmatrix} \qquad B_2^{-1} = \begin{pmatrix} 0 & \frac{1}{4} & 0 \\ -\frac{1}{2} & \frac{5}{8} & 0 \\ -\frac{1}{2} & \frac{23}{8} & 1 \end{pmatrix}$$

$$x_2^* = B_2^{-1}b = \begin{pmatrix} \frac{3}{4} \\ \frac{3}{8} \\ \frac{129}{8} \end{pmatrix} \quad z_2^* = c_{B_2}^T x_2^* = \begin{pmatrix} -7 & 8 & 0 \end{pmatrix} \begin{pmatrix} \frac{3}{4} \\ \frac{3}{8} \\ \frac{129}{8} \end{pmatrix} = -\frac{9}{4}$$

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_7 puesto que al hacer la

diferencia con su coeficiente en la función objetivo resultará siempre un valor positivo arbitrariamente grande, tal y como se advirtió ya en el paso anterior. En primer lugar, se muestra el cálculo de los vectores columna $y_i = B_2^{-1}a_i$:

$$y_2{=}B_2^{-1}a_2{=}\begin{pmatrix} -\frac{9}{4} \\ -\frac{61}{8} \\ -\frac{223}{8} \end{pmatrix}$$
$$y_4{=}B_2^{-1}a_4{=}\begin{pmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$$
$$y_5{=}B_2^{-1}a_5{=}\begin{pmatrix} \frac{1}{4} \\ \frac{5}{8} \\ \frac{23}{8} \end{pmatrix}$$

de modo que los costes reducidos de las variables no básicas son:

$$\begin{aligned} z_2 - c_2{=}c_{B_2}^T B_2^{-1}a_2 - c_2{=}\begin{pmatrix} -7 & 8 & 0 \end{pmatrix} y_2 + 3{=}-\tfrac{169}{4} \\ z_4 - c_4{=}c_{B_2}^T B_2^{-1}a_4 - c_4{=}\begin{pmatrix} -7 & 8 & 0 \end{pmatrix} y_4 + 0{=}4 \\ z_5 - c_5{=}c_{B_2}^T B_2^{-1}a_5 - c_5{=}\begin{pmatrix} -7 & 8 & 0 \end{pmatrix} y_5 + 0{=}\tfrac{13}{4} \end{aligned}$$

de donde resulta que la variable x_2 debe entrar en la base, puesto que es la única que tiene un coste reducido negativo.

c) Selección de la variable de salida
El paso anterior mostraba el cálculo de todos los vectores $\mathbf{y}$, de modo que a continuación se usan los valores de y_2 en el denominador de la regla de salida, puesto que la variable elegida para entrar es x_2:

$$\text{mín}\left\{ \frac{\frac{3}{4}}{\cancel{-\frac{9}{4}}}, \frac{\frac{3}{8}}{\cancel{-\frac{61}{8}}}, \frac{\frac{129}{8}}{\cancel{-\frac{223}{8}}} \right\}$$

donde, como puede verse, no hay ningún denominador estrictamente positivo. Tal y como se advertía al inicio de esta sección, ello indica que esta tarea de Programación Lineal no está acotada.

4. Para la resolución del último apartado, basta con recordar la *interpretación económica* de las soluciones de un problema dual que advierte que:

> La variable dual $x_i'^*$ indica la contribución por unidad del recurso i-ésimo b_i a la variación el en valor óptimo z^* actual del objetivo

Puesto que el enunciado requiere la contribución unitaria de todos los recursos, entonces es preciso calcular la solución completa del problema dual.

Para ello, es posible iniciar la aplicación de otro SIMPLEX. Sin embargo, en su lugar es preferible hacer uso del siguiente resultado teórico:

> Si el problema de programación lineal en forma simétrica tiene una solución óptima correspondiente a una base $\mathbf{B}$, entonces $\mathbf{x'}^T = \mathbf{c_B}^T\mathbf{B}^{-1}$ es una solución óptima para el problema dual

En este teorema, los términos usados para el cálculo de la solución óptima del problema dual se refieren al problema primal, salvo que se indique explícitamente lo contrario. Por lo tanto c_B^T es el vector de costes de las variables básicas en la solución del problema primal y B la base usada para el cálculo de la misma solución —que se mostró en el último paso de aplicación del SIMPLEX. Por el contrario, x'^T es la solución del problema dual.

En particular:

$$x'^* = c_B^T B^{-1} = (-7 \quad 8 \quad 0) \begin{pmatrix} 0 & \frac{1}{4} & 0 \\ -\frac{1}{2} & \frac{5}{8} & 0 \\ -\frac{1}{2} & \frac{23}{8} & 1 \end{pmatrix} = \begin{pmatrix} -4 & \frac{13}{4} & 0 \end{pmatrix}$$

de donde resulta que la contribución del primer recurso al crecimiento de la función objetivo es -4, mientras que el del segundo recurso es de $\frac{13}{4}$ por unidad de recurso. Nótese que:

- Primero, la contribución al crecimiento de la función objetivo de la primera restricción es un valor negativo. El motivo es que esta restricción estaba acotada inferiormente. Por lo tanto, aumentar una cota inferior deteriora necesariamente el valor de la función objetivo.
- Segundo, la contribución del tercer recurso es nulo y es que, como se explica en la siguiente sección, este es un recurso sobrante.

5. 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.
   - El valor de la función objetivo no está acotado puesto que los denominadores calculados en el tercer paso de la última iteración son todos negativos.
   - Por lo tanto, no existe una solución óptima a la tarea de Programación Lineal propuesta, puesto que existen siempre una cantidad infinita de puntos de la región factible que mejorarán el valor de la función objetivo de cualquier otro.

   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.

   En cuanto a las variables de holgura, se observa que hay una variable de holgura, x_6 en la última base que toma un valor positivo. Puesto que se trata de una variable añadida con signo positivo, entonces los recursos de la tercera restricción (esto es, a la que afecta esta variable de holgura), sobran.

   Por otra parte, en la sección anterior se ha calculado la solución óptima del problema dual que, como allí se indica, determina la contribución por unidad de cada recurso al crecimiento de la función objetivo. Tal y como allí ya se advirtió: el crecimiento del primer recurso deteriora el valor de la función objetivo, puesto que la primera restricción está acotada inferiormente, y puesto que hay recursos sobrantes en la tercera restricción, sumar más no hará crecer la función objetivo. Por lo tanto, la única política de inversión razonable consiste en hacer crecer los recursos de la segunda restricción que, de hecho, harán que la función objetivo crezca arbitrariamente puesto que, como se ha demostrado, esta tarea de Programación Lineal no está acotada.

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 simplemente con una ordenación particular de discos. A continuación se presentan definiciones estructuradas para cada uno de estos conceptos:

   JukeBox Cada configuración del JukeBox, u ordenación de discos, está caracterizada por una permutación `perm` de n posiciones, donde `perm [i]` representa el disco que ocupa la posición i-ésima. Esta representación asume, naturalmente, que cada disco está identificado de forma única, por ejemplo, con un número entero entre 0 y $(n-1)$.

   Por lo tanto, cualquier ordenación de discos es un estado diferente. En particular, el estado inicial se corresponde con la ordenación actual, y el estado final será el de la ordenación deseada.

   Acciones En este problema hay una única acción, `mover`, que toma como parámetros una posición entre 1 y $(n-1)$ `i`, y mueve el disco en la posición i-ésima a la primera, denotada como la posición 0.

   Basicamente, este operador no tiene precondiciones, de hecho, es posible mover cualquier disco en cualquier posición a la primera. Únicamente, para evitar redundancias y, también, para preparar el cálculo del factor de ramificación que se hará más tarde, se advierte que la única precondición es que el disco a mover debe estar entre la segunda y n-ésima posición, tal y como advertía el enunciado.

   Las postcondiciones describen el efecto de aplicar el operador que, en este caso, consiste en desplazar todas las posiciones del vector `perm` de las posiciones `0:i-1` a las posiciones `1:i`, e insertar el contenido anterior de `perm[i]` en la posición `0`.

   Por último, es importante determinar el coste del único operador identificado. Para ello, se observa que el enunciado pide explícitamente minimizar el número de movimientos del brazo de robot. Por lo tanto, independientemente de la posición del disco que se mueva, cada movimiento cuenta como una unidad y, por lo tanto, este será el coste del operador `mover`.

2. Tal y como se indicaba en el apartado anterior, cada ordenación de discos diferente es un estado distinto. Como quiera que con n discos hay hasta $n!$ ordenaciones diferentes, este es precisamente el tamaño del espacio de estados.

3. El primer apartado advertía que, en preparación al cálculo del factor de ramificación, la única precondición del operador `mover` es que $1 \leq \texttt{i} < n$. Por lo tanto, en cada paso, es posible mover hasta $(n-1)$ discos y, de hecho, no se pueden mover más de $(n-1)$ discos. Por lo tanto, este es el factor de ramificación mínimo y máximo.

4. Con un factor de ramificación tan alto como $(n-1)$, y un espacio de estados de tamaño factorial, los algoritmos del primero en amplitud son impracticables puesto que consumirían la capacidad del ordenador más potente con valores muy pequeños de n, $n < 20$. Para enfatizar esta observación, se observa que si la caracterización de cada estado consumiese sólo 1 byte, harían falta $2{,}265_1816{,}562$ Gigabytes para almacenar todos los estados que se pueden engendrar con $n = 20$.

   Por lo tanto, para resolver el problema óptimamente sólo se pueden considerar o el algoritmo del *primero en profundización iterativa* o el algoritmo de *ramificación y acotación en profundidad*:

Primero en profundización iterativa Inicialmente se invoca el algoritmo del primero en profundidad con un umbral igual a la unidad, de modo que se enumeran todas las permutaciones u ordenaciones de discos que se pueden hacer con un solo movimiento. Si algún nodo terminal generado de esta manera replica la ordenación deseada, entonces el algoritmo acaba con la solución óptima. En otro caso, se incrementa el umbral de la siguiente iteración en otra unidad. Necesariamente el algoritmo debe terminar en un número finito de iteraciones puesto que cualquier ordenación es alcanzable desde cualquier otra con el operador `mover` propuesto en la formalización del espacio de estados.

La ventaja de este algoritmo es que tiene un consumo de memoria lineal. Sin embargo, debe re-expandir todas las *transposiciones*.

Ramificación y acotación en profundidad El algoritmo de ramificación y acotación en profundidad aplicaría el algoritmo del primero en profundidad con un umbral de profundidad máxima igual a $+\infty$. Esto no provocará ningún error puesto que necesariamente se debe encontrar una solución inicial, y es que el espacio de estados de este problema es finito. A partir de ahí, utiliza el coste de la solución encontrada tentativamente para podar otras alternativas.

El algoritmo de ramificación y acotación en profundidad también tiene un consumo de memoria lineal, y como el algoritmo anterior, también debe re-expandir todas las *transposiciones*.

Cualquiera de estos algoritmos podría resolver el problema óptimamente, pero el primero resulta más adecuado puesto que: primero, el algoritmo de ramificación y acotación en profundidad se emplea normalmente en la presencia de costes; segundo, cuando los costes son unitarios, este algoritmo tiene un buen rendimiento si todas las soluciones se encuentran a la misma profundidad. Lo cierto es que ninguna de estas dos condiciones se satisfacen en el problema considerado y, por lo tanto, se recomienda primero en profundización iterativa.

5. Una heurística muy sencilla que puede obtenerse con la técnica de *relajación de restricciones*, consiste en relajar la restriccion de que el disco que se extrae de la pila debe colocarse el primero. En su lugar, se supone que puede colocarse en cualquier posición.

   Esto no significa, sin embargo, que el número de discos mal dispuestos sea una heurística admisible. De hecho, el número de discos mal dispuestos puede exceder el coste de resolver óptimamente una configuración determinada. Considérese, por ejemplo, la siguiente permutación de discos $\langle 0, 1, 4, 2, 3\rangle$ donde los tres últimos están mal dispuestos[2]. Sin embargo, con una sola aplicación del operador relajado (mover el disco 4 a la última posición), es posible ordenarlos todos resultando $\langle 0, 1, 2, 3, 4\rangle$. Asimismo, considérese ahora la permutación $\langle 1, 2, 3, 4, 0\rangle$, donde hay hasta 4 discos mal dispuestos, pero este problema puede resolverse óptimamente con una sola aplicación del operador *real*, esto es, $h^* = 1$.

   Se define a continuación el concepto de *inversión*. Se dice que las posiciones i y j están invertidas, con $j > i$, si y sólo si `permi[i]` debe disponerse después de `perm[j]` en la configuración final.

   Ahora sí, relajando la restricción indicada anteriormente, una heurística admisible (y bastante bien informada, por cierto) es el número de inversiones de la permutación actual que deben contabilizarse de la siguiente manera:

   - Si una posición i está invertida con una o más posiciones posteriores, se cuenta una única inversión, la del disco que hay en la posición i que debe *retrasarse* en la permutación.

     Por ejemplo, en el caso de la permutación $\langle 0, 1, 4, 2, 3\rangle$, hay una única inversión, el disco 4, que debe moverse al final de la permutación y, por lo tanto, la heurística devolvería 1 —en vez de 3, el número de discos mal dispuestos.
   - Si una posición i está invertida con una o más posiciones anteriores, se cuenta igualmente una única inversión, la del disco que hay en la posición i que debe *anticiparse* en la permutación.

     Considérse ahora el estado $\langle 1, 2, 3, 4, 0\rangle$. Con la regla indicada en este paso, se observa que sólo hay una inversión, la del disco 0 con todos los precedentes, que coincide con el valor óptimo para resolver este problema, puesto que con anticiparlo a la primera se recrea la configuración deseada.

[2] En todos los ejemplos de esta pregunta se asume que la ordenación deseada es $\langle 0, 1, 2, 3, 4\rangle$

6. La selección de un algoritmo de búsqueda informada para este caso depende, como en el caso de los algoritmos de búsqueda de fuerza bruta, 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 en el análisis de los algoritmos de fuerza bruta, si los algoritmos del primero en amplitud agotaban allí toda la memoria disponible para valores pequeños de n, en el caso de los algoritmos de búsqueda heurística ocurrirá lo mismo con los algoritmos del mejor primero. Por lo tanto, se desestima el uso del algoritmo A* y, en su lugar, se propone el segundo, IDA*.