Resolución de problemas usando lógica proposicional

Sesión 9

Edgar Andrade, PhD

Última revisión: Enero de 2020

Matemáticas Aplicadas y Ciencias de la Computación

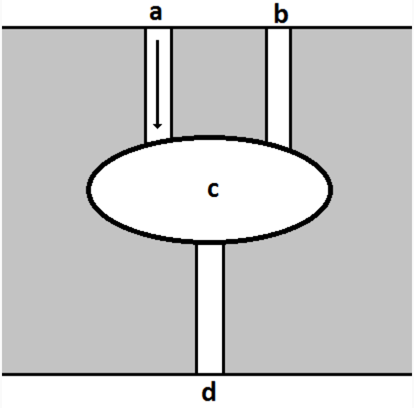
Presentación

En esta sesión estudiaremos:

1. Representación de situaciones sin condiciones iniciales
2. Representación de situaciones con condiciones iniciales
3. Búsqueda de soluciones
4. Interpretación de soluciones

Contenido

Problema —sin condiciones iniciales—



El problema es el de encontrar todas las rutas para un caminante que desea pasar de una orilla a otra.

Claves de representación

Cada letra proposicional representa cruzar un puente en una dirección:

p: Caminante va de a a c

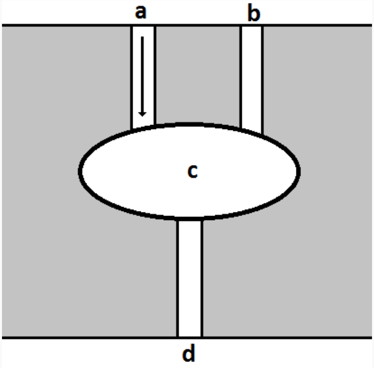
q: Caminante va de b a c

r: Caminante va de d a c

s: Caminante va de c a a

t: Caminante va de c a b

u: Caminante va de c a d



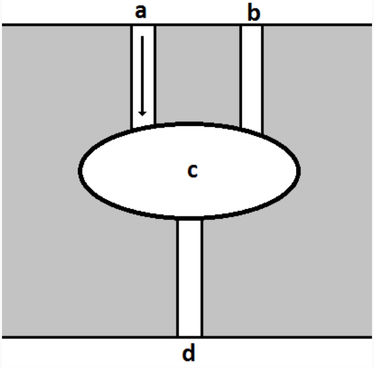
Tipos de reglas

Regla 1: Debe comenzar en una de las orillas.

Regla 2: Debe llegar a la otra orilla.

Regla 3: No comienza ni termina en la misma orilla.

Regla 1

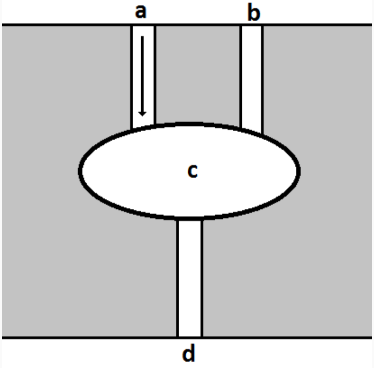


Comienza en a**:** $(p \wedge \neg q \wedge \neg r)$

Comienza en b**:** $(q \wedge \neg p \wedge \neg r)$

Comienza en d**:** $(r \wedge \neg p \wedge \neg q)$

Regla 1



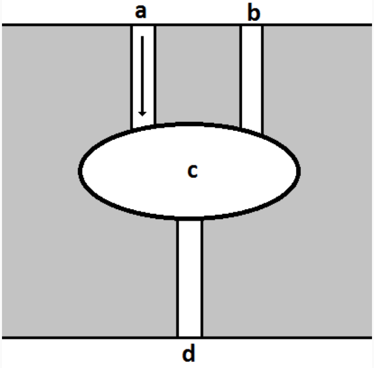
Comienza en a**:** $(p \wedge \neg q \wedge \neg r)$

Comienza en b**:** $(q \wedge \neg p \wedge \neg r)$

Comienza en d**:** $(r \wedge \neg p \wedge \neg q)$

Regla 1:$(p \wedge \neg q \wedge \neg r) \vee (q \wedge \neg p \wedge \neg r) \vee (r \wedge \neg p \wedge \neg q)$

Regla 2

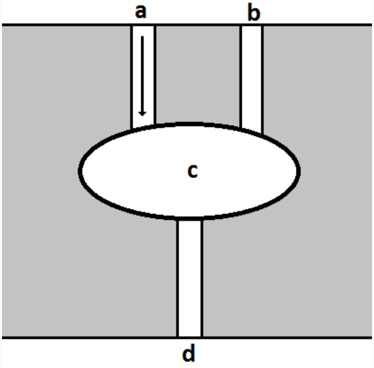


Termina en a**:** $(s \wedge \neg t \wedge \neg u)$

Termina en b**:** $(t \wedge \neg s \wedge \neg u)$

Termina en d**:** $(u \wedge \neg s \wedge \neg t)$

Regla 2



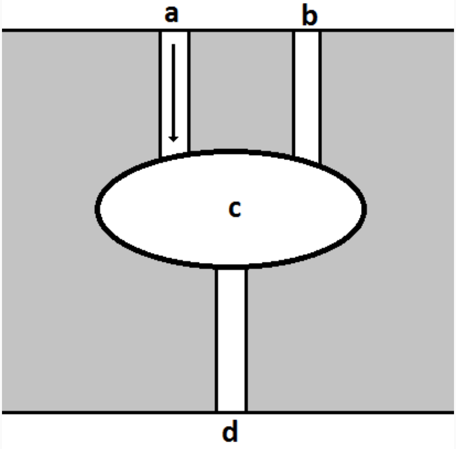
Termina en a**:** $(s \wedge \neg t \wedge \neg u)$

Termina en b**:** $(t \wedge \neg s \wedge \neg u)$

Termina en d**:** $(u \wedge \neg s \wedge \neg t)$

Regla 2:$(s \wedge \neg t \wedge \neg u) \vee (t \wedge \neg s \wedge \neg u) \vee (u \wedge \neg s \wedge \neg t)$

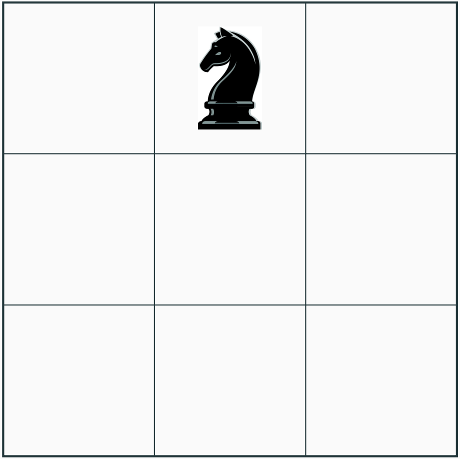
Regla 2



Regla 3:$(p \leftrightarrow \neg s) \wedge (q \leftrightarrow \neg t) \wedge (r \leftrightarrow \neg u)$

Contenido

Problema —con condiciones iniciales—



Dado un caballo en un tablero de ajedrez de 3×3, el problema consiste en ubicar otros dos caballos de tal manera que ningún caballo ataque a otro.

Tipos de reglas

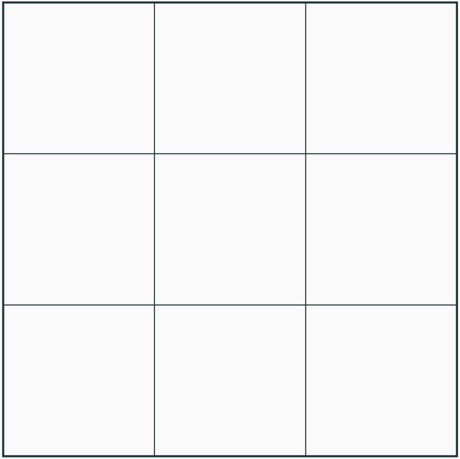
Regla 1: Debe haber exactamente tres caballos en el tablero.

Regla 2: Ningún caballo debe poder atacar a otro.

Regla 3: Debe haber un caballo en la casilla c_i.

Contenido

Más de dos opciones en una rejilla



El problema es poner todos los números del 1 al 9 en la rejilla, uno por casilla.

Claves de representación

Cada letra proposicional representa que un número está en una casilla:

p_1: El número 1 está en la casilla 1.
p_2: El número 1 está en la casilla 2.
$\vdots$
q_1: El número 2 está en la casilla 1.
q_2: El número 2 está en la casilla 2.
$\vdots$
x_1: El número 9 está en la casilla 1.
x_2: El número 9 está en la casilla 2.
$\vdots$
x_9: El número 9 está en la casilla 9.

Codificación usando Python (1/4)

```
def codifica(f, c):
    # Funcion que codifica la fila f y columna c
    if ((f < 1) or (f > Nf)):
        print('Fila incorrecta! Debe ser un numero entre 1 y', Nf)
        return None
    elif ((c < 1) or (c > Nc)):
        print('Columna incorrecta! Debe ser un numero entre 1 y', Nc)
        return None
    else:
        n = Nf*(f-1) + c
        return chr(255 + n)
```

Codificación usando Python (2/4)

```
def decodifica(x, Nf, Nc):
    # Funcion que codifica un caracter en su respectiva
    # fila f y columna c de la tabla
    n = ord(x) - 255
    if ((n < 1) or (n > Nf * Nc)):
        print('Caracter incorrecto! Debe estar entre 1 y', Nf * Nc)
        return None
    else:
        f = int(n / Nf) + 1
        c = n % Nf
        return f, c
```

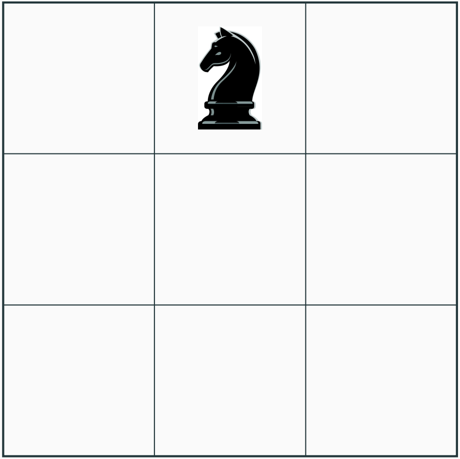
Codificación usando Python (3/4)

```
# Codificacion y decodificacion de una tabla
# de Nf filas y Nc columnas
Nf = 9 # Numero de filas
Nc = 9 # Numero de columnas
letrasProposicionales = [chr(i) for i in range(256, 256 + Nf*Nc)]
print('letrasProposicionales')
for i in range(Nc):
    print(letrasProposicionales[Nf*(i):Nf*(i+1)])
```

Codificación usando Python (4/4)

```
# Intente con varias opciones para fila y columna
fila = 9
columna = 5
print('La fila es', fila)
print('La columna es', columna)
n = codifica(fila, columna)
print('La codificacion es', n)
f, c = decodifica(n, Nf, Nc)
print('La decodificacion es fila', f, 'columna', c)
```

Dos opciones en una rejilla por cada paso



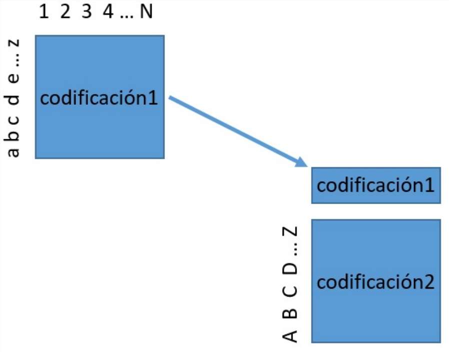
El caballo debe recorrer todo el tablero de ajedrez de 3×3, excepto la casilla central.

Claves de representación

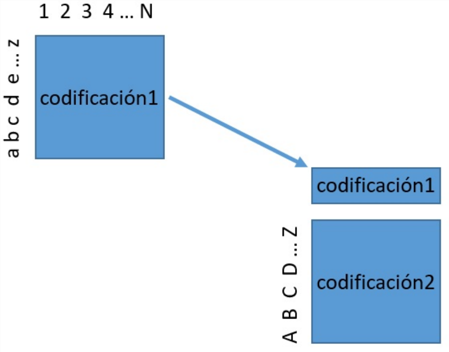
Cada letra proposicional representa que en un turno el caballo está en una casilla:

p_1: El caballo está en la casilla 1 en el turno 1.
p_2: El caballo está en la casilla 1 en el turno 2.
$\vdots$
q_1: El caballo está en la casilla 2 en el turno 1.
q_2: El caballo está en la casilla 2 en el turno 2.
$\vdots$
x_1: El caballo está en la casilla 9 en el turno 1.
x_2: El caballo está en la casilla 9 en el turno 1.
$\vdots$
x_9: El caballo está en la casilla 9 en el turno 9.

Más de dos opciones en una rejilla por cada paso

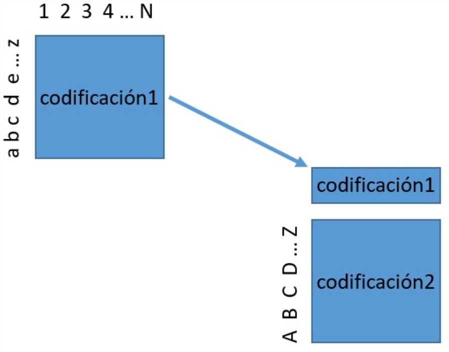


Más de dos opciones en una rejilla por cada paso



1, 2, 3, . . . representan las rejillas

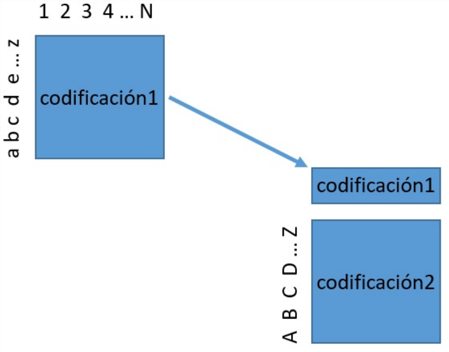
Más de dos opciones en una rejilla por cada paso



1, 2, 3, ... representan las rejillas

a, b, c, ... representan las opciones

Más de dos opciones en una rejilla por cada paso



1, 2, 3, . . . representan las rejillas

a, b, c, . . . representan las opciones

A, B, C, . . . representan los pasos

Contenido

Idea del procedimiento

1. Representar las reglas mediante fórmulas de la lógica proposicional, $\varphi_1, \ldots, \varphi_n$.

Idea del procedimiento

1. Representar las reglas mediante fórmulas de la lógica proposicional, $\varphi_1, \ldots, \varphi_n$.
2. Encontrar las interpretaciones I que hacen verdadera a la fórmula $\varphi_1 \wedge \ldots \wedge \varphi_n$.

Idea del procedimiento

1. Representar las reglas mediante fórmulas de la lógica proposicional, $\varphi_1, \ldots, \varphi_n$.
2. Encontrar las interpretaciones I que hacen verdadera a la fórmula $\varphi_1 \wedge \ldots \wedge \varphi_n$.

☞ Este procedimiento se puede realizar, aunque de manera muy ineficiente, mediante tablas de verdad.

Idea del procedimiento

1. Representar las reglas mediante fórmulas de la lógica proposicional, $\varphi_1, \ldots, \varphi_n$.
2. Encontrar las interpretaciones I que hacen verdadera a la fórmula $\varphi_1 \wedge \ldots \wedge \varphi_n$.

☞ Este procedimiento se puede realizar, aunque de manera muy ineficiente, mediante tablas de verdad.

3. Finalmente, las interpretaciones I encontradas se interpretan como soluciones del problema.

Idea del procedimiento —1—

1. Representar las reglas mediante fórmulas de la lógica proposicional

1a. Representar las reglas como cadenas en notación polaca inversa.

Idea del procedimiento —1—

1. Representar las reglas mediante fórmulas de la lógica proposicional

1a. Representar las reglas como cadenas en notación polaca inversa.

1b. Transformar las cadenas en árboles.

Intermezzo: Notación Polaca y Notación Polaca Inversa

La notación polaca pone los conectivos lógicos primero, seguidos de sus argumentos (no requiere paréntesis):

$p \to q$ se denota como $\to pq$

Intermezzo: Notación Polaca y Notación Polaca Inversa

La notación polaca pone los conectivos lógicos primero, seguidos de sus argumentos (no requiere paréntesis):

$p \to q$ se denota como $\to pq$

$p \land \neg(q \lor r)$ se denota como $\land p \neg \lor qr$

Intermezzo: Notación Polaca y Notación Polaca Inversa

La notación polaca pone los conectivos lógicos primero, seguidos de sus argumentos (no requiere paréntesis):

$p \to q$ se denota como $\to pq$ Inversa: $qp \to$

$p \land \neg(q \lor r)$ se denota como $\land p \neg \lor qr$

Intermezzo: Notación Polaca y Notación Polaca Inversa

La notación polaca pone los conectivos lógicos primero, seguidos de sus argumentos (no requiere paréntesis):

$p \to q$ se denota como $\to pq$ Inversa: $qp \to$

$p \land \neg(q \lor r)$ se denota como $\land p \neg \lor qr$ Inversa: $rq \lor \neg p \land$

1a

1a. Representar las reglas en notación polaca inversa:

Hay exactamente tres caballos en la primera fila.

1a

1a. Representar las reglas en notación polaca inversa:

Hay exactamente tres caballos en la primera fila.

$(((((((c_1 \wedge c_2) \wedge c_3) \wedge \neg c_4) \wedge \neg c_5) \wedge \neg c_6) \wedge \neg c_7) \wedge \neg c_8) \wedge \neg c_9$

1a

1a. Representar las reglas en notación polaca inversa:

Hay exactamente tres caballos en la primera fila.

$(((((((c_1 \wedge c_2) \wedge c_3) \wedge \neg c_4) \wedge \neg c_5) \wedge \neg c_6) \wedge \neg c_7) \wedge \neg c_8) \wedge \neg c_9$

Notación Polaca: $\wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge\, c_1 c_2 c_3 \neg c_4 \neg c_5 \neg c_6 \neg c_7 \neg c_8 \neg c_9$

1a

1a. Representar las reglas en notación polaca inversa:

Hay exactamente tres caballos en la primera fila.

$(((((((c_1 \land c_2) \land c_3) \land \neg c_4) \land \neg c_5) \land \neg c_6) \land \neg c_7) \land \neg c_8) \land \neg c_9$

Notación Polaca: $\land \land \land \land \land \land \land \land c_1 c_2 c_3 \neg c_4 \neg c_5 \neg c_6 \neg c_7 \neg c_8 \neg c_9$

NP Inversa: $c_9 \neg c_8 \neg c_7 \neg c_6 \neg c_5 \neg c_4 \neg c_3 c_2 c_1 \land \land \land \land \land \land \land \land$

1a

1a. Representar las reglas en notación polaca inversa:

Hay exactamente tres caballos en la primera fila.

$(((((((c_1 \wedge c_2) \wedge c_3) \wedge \neg c_4) \wedge \neg c_5) \wedge \neg c_6) \wedge \neg c_7) \wedge \neg c_8) \wedge \neg c_9$

Notación Polaca: $\wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge\, c_1 c_2 c_3 \neg c_4 \neg c_5 \neg c_6 \neg c_7 \neg c_8 \neg c_9$

NP Inversa: $c_9 \neg c_8 \neg c_7 \neg c_6 \neg c_5 \neg c_4 \neg c_3 c_2 c_1 \wedge \wedge \wedge \wedge \wedge \wedge \wedge \wedge$

☞ Faltan las otras $9 \times 8 \times 7 - 1$ reglas.

1a —Python—

```
# Regla 1: Debe haber exactamente tres caballos
letrasProposicionales = [str(i) for i in range(1, 10)] # Creo las letras proposicionales
conjunciones = ' ' # Para ir guardando las conjunciones de trios de disyunciones de literales
inicial = True # Para inicializar la primera conjuncion

for p in letrasProposicionales:
    aux1 = [x for x in letrasProposicionales if x != p] # Todas las letras excepto p
    for q in aux1:
        aux2 = [x for x in aux1 if x != q] # Todas las letras excepto p y q
        for r in aux2:
            literal = r + q + p + 'Y' + 'Y'
            aux3 = [x + '-' for x in aux2 if x != r]
            for k in aux3:
                literal = k + literal + 'Y'
            if inicial: # Inicializar la primera conjuncion
                conjunciones = literal
                inicial = False
            else:
                conjunciones = literal + conjunciones + 'O'
```

1b —Python—

Transformar las cadenas en árboles.

```
def string2Tree(A, letrasProposicionales):
    # Crea una formula como tree dada una formula como cadena escrita en notacion polaca inversa
    # Input: A, lista de caracteres con una formula escrita en notacion polaca inversa
    #         letrasProposicionales, lista de letras proposicionales
    # Output: formula como tree
    conectivos = [’O’, ’Y’, ’>’]
    pila = []
    for c in A:
        if c in letrasProposicionales:
            pila.append(Tree(c, None, None))
        elif c == ’-’:
            formulaAux = Tree(c, None, pila[-1])
            del pila[-1]
            pila.append(formulaAux)
        elif c in conectivos:
            formulaAux = Tree(c, pila[-1], pila[-2])
            del pila[-1]
            del pila[-1]
            pila.append(formulaAux)
    return pila[-1]
```

Idea del procedimiento —2—

2. Encontrar las interpretaciones I que hacen verdadera a la fórmula $\varphi_1 \wedge \ldots \wedge \varphi_n$.

   2a. Construir todas las interpretaciones.

Idea del procedimiento —2—

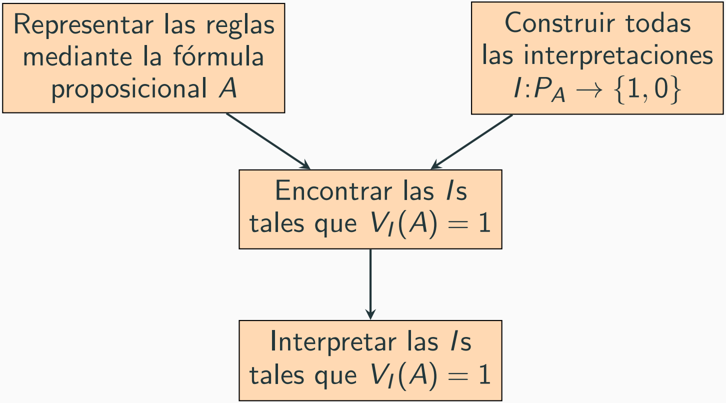
2. Encontrar las interpretaciones I que hacen verdadera a la fórmula $\varphi_1 \wedge \ldots \wedge \varphi_n$.

2a. Construir todas las interpretaciones.

2b. Para cada I, determinar si $V_I(\varphi_1 \wedge \ldots \wedge \varphi_n) = 1$.

Contenido

Esquema del procedimiento



Interpretación de soluciones

Las soluciones son asignaciones de valores 0s y 1s a letras proposicionales, es decir interpretaciones Is, tales que $V_I(\varphi_1 \wedge \ldots \wedge \varphi_n) = 1$.

Interpretación de soluciones

Las soluciones son asignaciones de valores 0s y 1s a letras proposicionales, es decir interpretaciones Is, tales que $V_I(\varphi_1 \wedge \ldots \wedge \varphi_n) = 1$.

Para resolver el problema, es indispensable interpretar esos valores de las letras proposicionales en la situación representada.

Interpretación de soluciones

Las soluciones son asignaciones de valores 0s y 1s a letras proposicionales, es decir interpretaciones Is, tales que $V_I(\varphi_1 \wedge \ldots \wedge \varphi_n) = 1$.

Para resolver el problema, es indispensable interpretar esos valores de las letras proposicionales en la situación representada.

Por ejemplo, en el problema de los caballos, las letras proposicionales con valor 1 son las casillas donde van los caballos.

Interpretación de soluciones —ejemplo caballos—

$\{$1:0, 2:1, 3:0,
4:0, 5:0, 6:1,
7:0, 8:1, 9:0$\}$

$\{$1:0, 2:0, 3:0,
4:0, 5:1, 6:1,
7:0, 8:0, 9:1$\}$

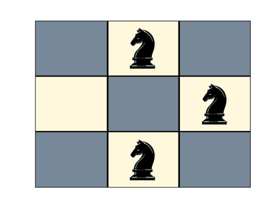
$\{$1:0, 2:0, 3:0,
4:0, 5:1, 6:0,
7:1, 8:1, 9:0$\}$

Interpretación de soluciones —ejemplo caballos—

{1:0, 2:1, 3:0,
4:0, 5:0, 6:1,
7:0, 8:1, 9:0}

{1:0, 2:0, 3:0,
4:0, 5:1, 6:1,
7:0, 8:0, 9:1}

{1:0, 2:0, 3:0,
4:0, 5:1, 6:0,
7:1, 8:1, 9:0}

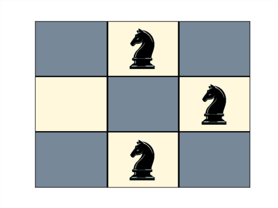


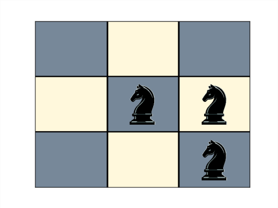
Interpretación de soluciones —ejemplo caballos—

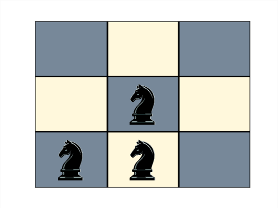
$\{1{:}0,\ 2{:}1,\ 3{:}0,\ 4{:}0,\ 5{:}0,\ 6{:}1,\ 7{:}0,\ 8{:}1,\ 9{:}0\}$

$\{1{:}0,\ 2{:}0,\ 3{:}0,\ 4{:}0,\ 5{:}1,\ 6{:}1,\ 7{:}0,\ 8{:}0,\ 9{:}1\}$

$\{1{:}0,\ 2{:}0,\ 3{:}0,\ 4{:}0,\ 5{:}1,\ 6{:}0,\ 7{:}1,\ 8{:}1,\ 9{:}0\}$





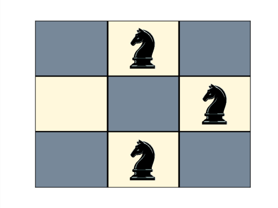


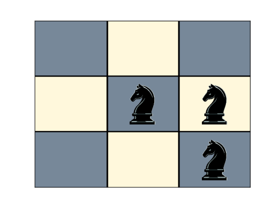
Interpretación de soluciones —ejemplo caballos—

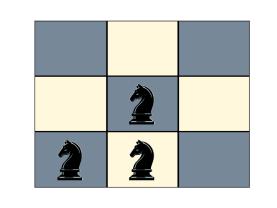
{1:0, 2:1, 3:0,
4:0, 5:0, 6:1,
7:0, 8:1, 9:0}

{1:0, 2:0, 3:0,
4:0, 5:1, 6:1,
7:0, 8:0, 9:1}

{1:0, 2:0, 3:0,
4:0, 5:1, 6:0,
7:1, 8:1, 9:0}







☞ Consultar archivo 'visualizacion_tablero.py'.

Ejercicio

Escribir un código python que dibuje todas las soluciones posibles al problema de poner tres caballos en un tablero de ajedrez de tamaño 3×3, sin que se ataquen mutuamente, dados dos caballos iniciales: uno en la casilla 2 y otro en la casilla 6. [¡Son cuatro soluciones posibles!]

Fin de la sesión 7

En esta sesión usted ha aprendido a:

1. Representar situaciones mediante la lógica proposicional, con y sin condiciones iniciales.
2. Buscar soluciones mediante tablas de verdad.
3. Interpretar las soluciones.