PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
Facultad de Ingeniería
IIC2613 Inteligencia Artificial
2024-1

Tarea 2

Programación Lógica para el Razonamiento

Fecha de entrega: martes 16 de abril a las 23:59 hrs

Aspectos generales

Formato y plazo de entrega

El formato de entrega son archivos con extensión `.lp` con un PDF para las respuestas teóricas. El lugar de entrega es en el repositorio de la tarea, en la branch por defecto, hasta el martes 16 de abril a las 23:59 hrs. Para crear tu repositorio, debes entrar en el enlace del anuncio de la tarea en Canvas. Por último, recuerda que los cupones de atraso son días **no hábiles** extra.

Integridad Académica

Este curso se adhiere al Código de Honor establecido por la universidad, el cual tienes el deber de conocer como estudiante. Todo el trabajo hecho en esta tarea debe ser **totalmente individual**. La idea es que te des el tiempo de aprender estos conceptos fundamentales, tanto para el curso, como para tu formación profesional. Las dudas se deben hacer exclusivamente al cuerpo docente a través de las *issues* en GitHub.

Por otra parte, sabemos que estás utilizando material hecho por otras personas, por lo que es importante reconocerlo de la forma apropiada. Todo lo que obtengas de internet debes citarlo de forma correcta (ya sea en APA, ICONTEC o IEEE). Cualquier falta a la ética y/o a la integridad académica será sancionada con la reprobación del curso y los antecedentes serán entregados a la Dirección de Pregrado.

Comentarios adicionales

El objetivo de esta tarea es que puedan desarrollar la capacidad de modelar problemas a partir del lenguaje natural y resolverlos utilizando ASP en Clingo. Es fundamental que pongan énfasis en las justificaciones de sus respuestas, cuidando la redacción, ortografía; manteniendo el código ordenado y comentado. Aquellas respuestas que solo presenten resultados o código (sin contexto ni comentarios) no serán consideradas, mientras que tareas desordenadas pueden ser objeto de descuentos.

Se recomienda fuertemente apoyarse de los apuntes del curso para el desarrollo de la tarea. En particular el uso de Clingo y modelación de problemas.

1. Reflexión y Teoría (2 pts.)

Mucha gente argumenta que el podcast de Lex Fridman en YouTube es imperdible para todo quien se dedique a la inteligencia artificial. Hace un par de semanas, Lex publicó una entrevista a Yann Lecun, ganador del premio Turing (el equivalente a un premio Nobel en Ciencias de la Computación) conocido como uno de los "padrinos del Deep Learning". Para contestar esta pregunta, primero ve la sección sobre alucinaciones y razonamiento en modelos de lenguaje en el siguiente video: `https://www.youtube.com/watch?v=5t1vTLU7s40&t=3967s` (1:06:08 - 1:29:28).

1. En la entrevista, Yann comenta sobre el fenómeno manifestado por modelos de lenguaje natural conocido como *alucinación*. ¿A qué se refiere con este concepto? Explica en tus propias palabras.

2. Minutos más tarde, los locutores comienzan a discutir respecto a la capacidad de razonamiento de los modelos de lenguaje. Yann argumenta que los modelos de lenguaje actuales **NO** son capaces de desempeñar razonamiento lógico debido a la manera en que son entrenados. ¿Estas de acuerdo con esta opinión? Justifica brevemente. ¿Como describirías la capacidad de razonar en tus propias palabras?

3. Contrario a los modelos de lenguaje, sistemas lógicos como Clingo nos permiten extraer conclusiones lógicas a partir de un problema bien definido con una serie de reglas, habilidad que se ha demostrado compleja para los grandes modelos de lenguaje. ¿A que crees que se debe esto? Haz referencia a la manera en que son entrenados y el concepto de alucinación.

4. Propone una idea (no debe ser una implementación concisa, basta con un concepto ambiguo) de como fomentar el razonamiento lógico en modelos de lenguaje para obtener conclusiones similares a las que genera Clingo pero de manera automática.

2. DCChess (2 pts.)

En este problema deberás resolver un juego basado en el ajedrez. El objetivo es que la pieza del jugador (en este caso, el rey) llegue a alguna casilla objetivo. Sin embargo, en el tablero hay piezas enemigas que pueden atacarla, por lo que deberás crear las reglas necesarias para que el esta pueda cumplir con su cometido sin ser atacada en el camino.

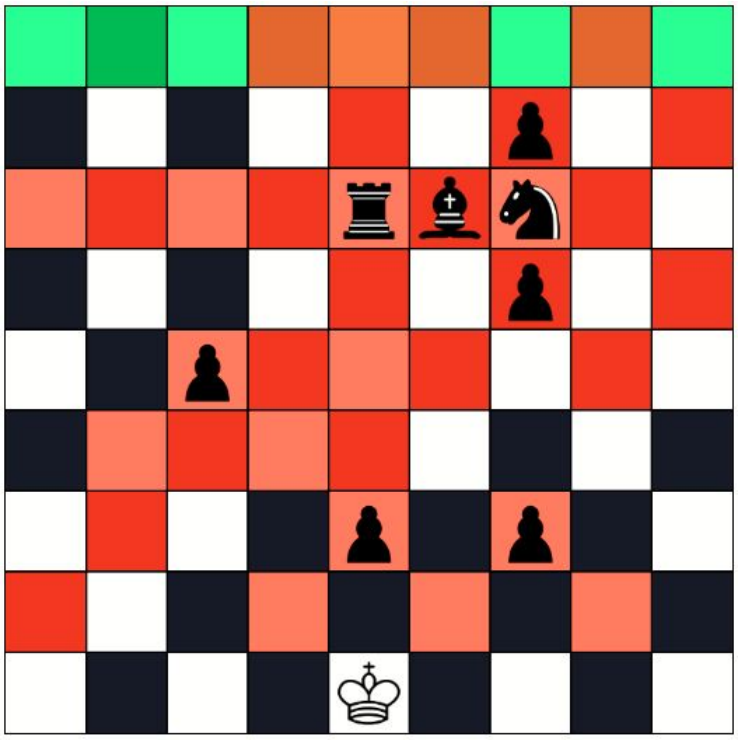


Figura 1: Posible mapa DCChess, jugador: king

Las posibles piezas atacantes son:

```
% Torre, ataca mediante líneas rectas horizontales y verticales
rook(X,Y,Id). % La torre representada por Id se encuentra ubicada en (X, Y)

% Alfil, ataca mediante líneas diagonales.
bishop(X,Y,Id). % El alfil representado por Id se encuentra ubicado en (X, Y)

% Caballo, sus posibilidades de ataque son en forma de "L".
knight(X,Y). % El caballo se encuentra ubicado en (X, Y)

% Peón, ataca solamente en sus diagonales delanteras (una casilla abajo en el eje y).
pawn(X,Y). % El peon se encuentra ubicado en (X, Y)
```

El rey se representa con la siguiente regla:

```
% Rey, se mueve en todas las direcciones, pero solo una casilla a la vez.
king(X,Y). % El rey se encuentra ubicado en (X, Y)
```

Las limitaciones del juego son las siguientes:

- El rey se mueve en todas las direcciones, incluyendo las diagonales, pero **solo una casilla a la vez**.
- El rey **no puede moverse a una casilla donde haya una pieza enemiga**.
- El rey **no puede moverse a una casilla donde pueda ser atacado**.
- Las piezas enemigas **no se pueden mover**, solo atacarán al rey si este se encuentra en su rango de ataque.

Se te entregará un mapa con las siguientes características:

- El mapa es de tamaño **NxN**.
- La posición (0,0) está dada en la esquina inferior izquierda y la posición (N-1,N-1) está dada en la esquina superior derecha (similar a un plano cartesiano).
- En el mapa habrán casillas vacías (blancas y negras), casillas con piezas, casillas bloqueadas por los ataques (rojas) y casillas objetivo (verdes).

Predicados

Los predicados que se te entregarán son los siguientes:

- Piezas

```
king(X,Y,T). % Representa la posición del rey en el tablero en el tiempo T.
rook(X,Y,Id). % Representa la posición de la torre con id=Id en el tablero.
bishop(X,Y,Id). % Representa la posición del alfil con id=Id en el tablero.
knight(X,Y). % Representa la posición de un caballo en el tablero.
pawn(X,Y). % Representa la posición de un peón en el tablero.
```

(Las piezas bishop y rook tienen un identificador Id que se utiliza para diferenciarlas en sus lineas de ataque, las demás piezas no lo necesitan.)

- Otros

```
kingAction(action). % Las opciones de movimiento del rey.
able(X,Y). % Posición no bloqueada.
exec(X,Y). % Movimiento a ejecutar.
pieceOn(X,Y,T). % Posición ocupada por la pieza jugadora en el tiempo T. En la
↪  parte 1 es el rey, en la parte 2 es el caballo.
goal(X,Y). % Posición objetivo.
done(T). % Indica que el juego ha terminado en el tiempo T.
piece(name). % Indica la pieza jugadora, que puede ser king o knight.
square(X,Y). % Casilla existente en el tablero.
time(T). % Tiempos disponibles en el juego, permite darle límite de cálculo a
↪  clingo.
blocked(X,Y). % Posición bloqueada por una pieza enemiga o su ataque.
rook_attack_d(X,Y,I)., _u, _r, _l % Lineas de ataque de la torre.
attack_line(X,Y,I). % Línea de ataque de la pieza I.
end_attack_line(X,Y,I). % Término de la línea de ataque de la pieza I.
```

Actividades

2.1. Actividad 1: Bloqueos para el camino del rey (1 pto.)

Para resolver este problema, deberás completar el archivo `solution.lp` con las reglas necesarias para que el rey `piece(king)`. pueda llegar a alguna casilla de la fila final sin ser atacado por ninguna pieza enemiga. Las reglas a implementar son las siguientes:

- **Bloqueo de las piezas enemigas:** Deberás implementar las reglas necesarias para que las piezas enemigas bloqueen las casillas que corresponden a su línea de ataque. En el archivo base `solution.lp` se encuentran las reglas para las piezas `pawn(X,Y)` (peón) y `rook(X,Y,I)` (torre). Deberás completar las reglas para las piezas `knight(X,Y)` (caballo) y `bishop(X,Y,I)` (alfil) **(0.5 ptos por pieza)**.

2.2. Actividad 2: Camino del caballo (1 pto.)

Para la segunda parte del problema, deberás implementar el juego para que la pieza jugadora sea un caballo `piece(knight)`. Deberás completar las reglas necesarias para que el caballo pueda llegar a alguna casilla de la fila final sin ser atacado por ninguna pieza enemiga. Las reglas a implementar son las siguientes:

- **Acciones del caballo:** Deberás implementar las acciones posibles del caballo como jugador, según sus movimientos en L (los mismos que en ajedrez) **(0.25 ptos)**.
- **Movimiento del caballo:** Deberás implementar las reglas necesarias para que el caballo pueda moverse a lo largo del tablero **(0.75 ptos)**.

El bloqueo de las piezas enemigas es el mismo que en la parte anterior, por lo que no es necesario implementar nuevas reglas para este caso.

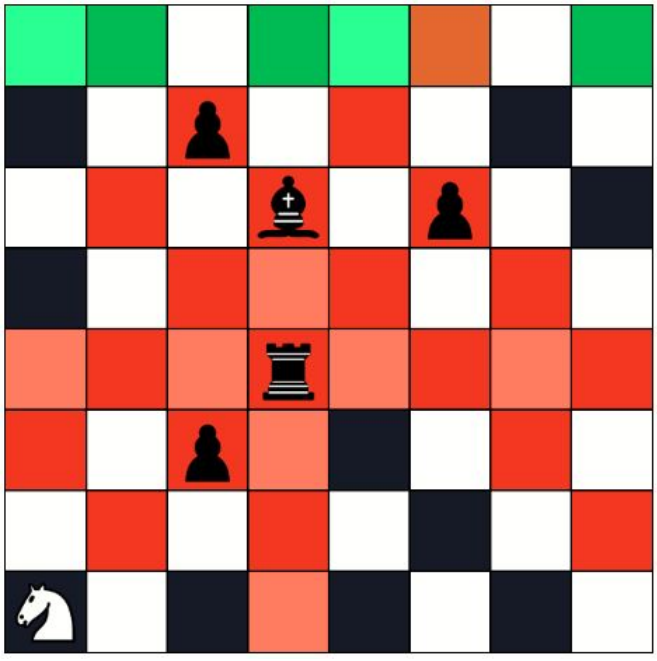


Figura 2: Posible mapa DCChess, jugador: knight

Carpetas y archivos

Carpetas

- `maps`: Contiene mapas de ejemplo a resolver. Dentro esta se encuentran 2 subcarpetas, **king** y **knight**, que contienen mapas para testear las partes 1 y 2 respectivamente.
- `visualizer`: Contiene la carpeta `imgs` con los sprites de la visualización y el archivo `DCChess.html` que se encarga de visualizar el mapa y la solución entregada.

Archivos

Los archivos que tendrás para resolver el problema son:

- `maps/piece/../mapX.lp`: Archivo con el mapa a resolver, donde X es el número del mapa y piece es king o knight. No los debes modificar.
- `solution.lp`: Archivo que contiene las reglas que solucionan los mapas. Acá debes completar las reglas para que resuelvan el problema.
- `process.py`: Archivo que contiene el código que se encarga de leer la solución entregada por clingo en la consola, parsearla y generar un archivo `output.txt`. No debes modificarlo.
- `visualizer/DCChess.html`: Archivo que contiene el visualizador del mapa junto con la solución entregada. Al abrirlo en un navegador se debe ingresar al botón "Seleccionar archivo" y cargar la solución entregada en `output.txt`. No lo debes modificar.

Comandos a utilizar

- Para comenzar a resolver el problema puede ser útil querer manipular solamente el archivo de clingo y ver la salida en la consola. Para esto se puede utilizar el siguiente comando:

  `clingo maps/king/../mapX.lp solution.lp`
  `clingo maps/knight/../mapX.lp solution.lp` (para jugar con el caballo)

  Donde `..` debe ser reemplazado según la subcarpeta dentro de king o knight. Con esto, se podrá ver la salida de clingo en la consola y verificar si las reglas están funcionando correctamente. En esta parte, puedes modificar los statements `#show` para que te sea más fácil depurar el código.

- Una vez que tengas una solución que te parezca correcta, puedes ejecutar el siguiente comando para generar el archivo `output.txt`:

  `clingo maps/piece/../mapX.lp solution.lp | python process.py`

  Este comando ejecutará clingo y luego procesará la salida para generar el archivo `output.txt`. Es muy importante que, en esta parte, mantengas los *statements* `#show` que se entregaron inicialmente en el archivo `solution.lp` para que el proceso de parseo funcione correctamente. Si el comando `python` no funciona, puedes intentar con `python3` o `py`.

- A continuacion se incluyen los *statements* `#show` inicialmente entregados en el archivo `.lp`:

```
% Statements visualizer
#show square/2.
#show time/1.
#show done/1.
#show goal/2.
#show bishop/3.
#show knight/2.
#show pawn/2.
#show rook/3.
#show pieceOn/3.
#show piece/1.
#show blocked/2.
```

- Finalmente, puedes abrir el archivo `DCChess.html` en un navegador y cargar el archivo `output.txt` que se generó en el paso anterior. De esta forma, podrás visualizar el mapa y la solución entregada por clingo.

Entrega

- Deberás entregar el archivo `solution.lp` con las reglas implementadas para resolver el problema.
- El archivo `solution.lp` debe estar correctamente comentado para que se entienda la lógica de las reglas implementadas.

3. DCCursos (2 pts.)

3.1. Introducción

La escuela de ingeniería destaca por tener una amplia variedad de especialidades para sus alumnos, quienes pueden elegir entre distintos programas de Major y ordenar sus semestres como prefieran. Sin embargo, esta flexibilidad conlleva también ciertas dificultades, pues los alumnos deben elegir con cuidado que ramos cursar en cada semestre, considerando restricciones como los prerrequisitos y correquisitos.

Debido a esto se te ha encomendado confeccionar un nuevo proyecto de Planificación de semestres, que ayudará a tus compañeros y generaciones futuras a organizar con facilidad sus semestres en ingeniería.

Figura 3: Planner de la escuela de ingeniería, DCCursos carece de interfaz gráfica

Los tests que se te entregarán son programas de Clingo que deben correr antes de tu programa. Estos podrían tener los siguientes predicados:

3.1.1. Cursos:

```
% Cantidad de cursos del Plan Común/Major por cursar, representan IDs para un mismo bloque
cursos_pc(0..M).    % cursos_pc(0), cursos_pc(1), (...), cursos_pc(M)
cursos_major(0..N). % cursos_major(0), cursos_major(1), (...), cursos_major(N)

% Curso del Plan Común/Major obligatorio y el ID del bloque al que pertenece
pc_necesario(Sigla, ID).    % Ej. pc_necesario("MAT1610", 0)
major_necesario(Sigla, ID). % Ej. major_necesario("IIC2613", 0)

% Curso del PC/Major e ID de su bloque. Pueden haber más optativos con el mismo ID,
↪  indicando así que son cursos equivalentes para avanzar en el PC/Major
pc_optativo(Sigla, ID).    % pc_optativo("FIS1514", 6) y pc_optativo("ICE1514", 6)
major_optativo(Sigla, ID).

% Curso de cualquier tipo y el semestre "S" en que se imparte.
curso(Sigla, S). % S=0 es ambos semestres, S=1 es solo impares y S=2 es solo pares
```

3.1.2. Requisitos:

```
% El curso C2 es prerrequisito para tomar el curso C1. Esto quiere decir que el curso C2
↪  debe planificarse en un semestre previo al semestre en que se toma C1.
requisito(C1, IDX1, IDX2, IDX3, C2).

% El curso C2 es correquisito para tomar el curso C1. Esto quiere decir que el curso C2
↪  debe planificarse en un semestre previo O IGUAL al semestre en que se toma C1.
correquisito(C1, IDX1, IDX2, IDX3, C2).
```

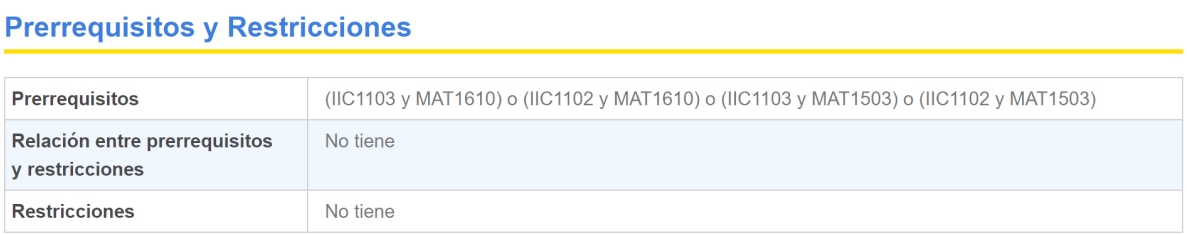
3.1.3. Subrequisitos:

Dentro del contexto de la tarea, llamaremos 'subrequisitos' a los predicados auxiliares utilizados para modelar si se cumplen los requisitos/correquisitos para poder tomar un ramo.

Corresponden a los índices `IDX1, IDX2, IDX3` en los predicados `requisito` y `correquisito`, pero no son un predicado en si mismos. Su objetivo es modelar las distintas relaciones entre requisitos, como conjunciones **"AND"** y disyunciones **"OR"**.

- **IDX1**: Necesitas cumplir con *al menos un* `IDX1` para tomar el ramo `C1`, es decir, entre distintos `IDX1` se tratan como un **"OR"**.
- **IDX2**: Necesitas cumplir con *todos* los `IDX2` correspondientes **a un mismo** `IDX1` para considerar ese `IDX1` como cumplido, es decir, entre distintos `IDX2` se tratan como un **"AND"**.
- **IDX3**: Necesitas cumplir con *al menos un* `IDX3` correspondiente **a un mismo** `IDX2` para considerar ese `IDX2` como cumplido, es decir, entre distintos `IDX3` se tratan como un **"OR"**.

El motivo para esta separacion es debido a la manera en que los requisitos de un curso son reportados dentro del Buscacursos UC:



Prerrequisitos y Restricciones

Prerrequisitos	(IIC1103 y MAT1610) o (IIC1102 y MAT1610) o (IIC1103 y MAT1503) o (IIC1102 y MAT1503)
Relación entre prerrequisitos y restricciones	No tiene
Restricciones	No tiene

Figura 4: Información del curso `IIC2115`, extraída directamente de Buscacursos UC

El curso `IIC2115`, será entonces representado por:

```
requisito("IIC2115", 0, 0, 0, "IIC1103").   % IIC1103 y MAT1610 comparten IDX1
requisito("IIC2115", 0, 1, 0, "MAT1610").   % ----------------------------
requisito("IIC2115", 1, 0, 0, "IIC1102").   % IIC1102 y MAT1610 comparten IDX1
requisito("IIC2115", 1, 1, 0, "MAT1610").   % ----------------------------
requisito("IIC2115", 2, 0, 0, "IIC1103").   % IIC1103 y MAT1503 comparten IDX1
requisito("IIC2115", 2, 1, 0, "MAT1503").   % ----------------------------
requisito("IIC2115", 3, 0, 0, "IIC1102").   % IIC1102 y MAT1503 comparten IDX1
requisito("IIC2115", 3, 1, 0, "MAT1503").   % ----------------------------
```

Un ejemplo paso a paso:
Supongamos que queremos tomar el ramo "Fundamentos de Teoría Electromagnética", de sigla `IEE1533`, entonces los siguientes predicados modelan sus requisitos y correquisitos:

```
% En palabras los requisitos son:
% FIS0153(c) y (MAT1630 o MAT1523 o MAT230E o MLM1130)

correquisito("IEE1533", 0, 0, 0, "FIS0153").
requisito("IEE1533", 0, 1, 0, "MAT1630").
requisito("IEE1533", 0, 1, 1, "MAT1523").
requisito("IEE1533", 0, 1, 2, "MAT230E").
requisito("IEE1533", 0, 1, 3, "MLM1130").
```

Analizemos cada subrequisito `IDX`:

- `IDX1`: Como todos los requisitos y correquisitos tienen un mismo `IDX1` igual a 0, entonces nos tenemos que fijar en el segundo índice, ya que para cumplir con el `IDX1` vamos a necesitar cumplir con todos los `IDX2` distintos.
- `IDX2`: Vemos que hay dos índices distintos para `IDX2`, 0 y 1 que queremos completar. Recordemos que los `IDX2` distintos se tratan como conjunciones, es decir, necesitamos elegir un curso de `IDX2` $= 0$ y un curso de `IDX2` $= 1$ para asi completar el subrequisito `IDX1`.
- `IDX3`: Por último, notamos que dentro de cada índice `IDX2`, no se repite ningún `IDX3`, es decir, podemos tomar cualquiera de los ramos disponibles así:
  - Para IDX2 $= 0$:
    Tenemos solo una opción y es
    - `FIS0153`
  - Para IDX2 $= 1$:
    Tenemos 4 opciones, debiendo cursar **al menos una** de ellas para cumplir con IDX2 $= 1$.
    - `MAT1630`
    - `MAT1523`
    - `MAT230E`
    - `MLM1130`

Finalmente, una posible solución (de 4 totales), sería cursar `FIS0153` y `MAT1630` antes de `IEE1533`

Carpetas y archivos

Carpetas

- `tests/`: Contiene test cases que deberás resolver utilizando Clingo, los cuales definen los cursos de major y plan común que te falta cursar junto con los requisitos para cada uno de ellos.

  - `tests/easy/`:
    Los mapas más fáciles con que probar, contienen ejemplos cortos y sencillos para probar la dependencia de requisitos/semestres pares e impares y cursos optativos.
  - `tests/medium/`:
    Mapas de mayor dificultad y extensión, contienen ejemplos para probar requisitos y correquisitos bajo un mismo IDX2 y el plan común completo.
  - `tests/hard/`:
    Planes de estudios reales para los Majors de Ingenieria (representados por su sigla), corresponden a datos reales extraídos de Buscacursos y SIDING.

Archivos

Los archivos que tendrás para resolver el problema son:

- `solution.lp`: Archivo que contiene las reglas que solucionan los tests. Acá debes completar las reglas para que resuelvan el problema.

Comandos a utilizar

Para probar tu código se puede utilizar el siguiente comando:

```
clingo solution.lp (PATH_TEST) -c bound=(VALOR)
```

Donde `bound=(VALOR)` representará el número de semestres en que se busca obtener el plan de estudios (variará para cada test).

Nota: Para evaluar tu programa, es obligatorio que crees el predicado `tomar(C, S)` que indica que el curso `C` (indicado por su sigla) se debe tomar en el semestre `S` (indicado por un número). Este predicado lo podrás visualizar utilizando el *statement* `#show`, para que te sea más fácil depurar el código.

```
#show tomar/2.
```

Entrega

- Deberás entregar el archivo `solution.lp` con las reglas implementadas para resolver el problema.
- El archivo `solution.lp` debe estar correctamente comentado para que se entienda la lógica de las reglas implementadas.

Actividades

3.2. Actividad 1: Reglas Básicas (1 pto.)

Lo primero que debes hacer es implementar las reglas más básicas que necesitará tu programa para funcionar.

- 1. Cada curso puede tomarse solo una vez
- 2. Cada semestre se toman a lo mas 6 cursos
- 3. Se toman todos los cursos `X_necesarios` (para el plan común y major)
- 4. Solo se puede tomar un curso en un semestre par/impar si es dictado en ese semestre
- 5. Todos los ID de bloques deben tener al menos uno de sus cursos inscritos (o bien se tomo el `X_necesario` o al menos uno de los `X_optativo`)

3.3. Actividad 2: Requisitos y Correquisitos (1 pto.)

- Implementar los subrequisitos exitosamente (Se recomienda seguir este orden):
  - **Subrequisito 3:** `req3(C, IDX1, IDX2)`
    Para un mismo IDX1 e IDX2 (y distinto IDX3)
    Alguno de los requisitos de mismo IDX1 e IDX2 fue tomado en un semestre anterior (anterior o igual en el caso de los correquisitos).
  - **Subrequisito 2:** `req2(C, IDX1)`
    Para un mismo IDX1 (y distinto IDX2)
    Se cumple el subrequisito 3 para **todos** los distintos IDX2 que comparten IDX1
  - **Subrequisito 1:** `req1(C)`
    Para un curso
    Se cumple con **al menos uno** de los subrequisitos 2 para el curso
- Si un curso tiene al menos un requisito/correquisito, solamente puede ser tomado si cumple el subrequisito 1