Trabajo Práctico 2 - Programación Lógica
Viajante de comercio

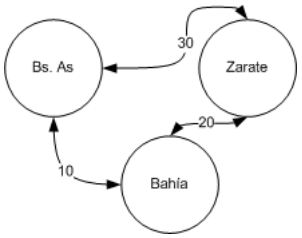
Fecha de entrega: jueves 30 de octubre, hasta las 21 hs.

1. Mapas y rutas

Una empresa de transporte desea organizar su logística. Para ello, está confeccionando un mapa con las rutas que realizan sus camiones.

Un mapa es un grafo en el que sus nodos representan ciudades y sus ejes distancias entre éstas. Este grafo está representado mediante una lista de adyacencia. Cada elemento de la lista es una ruta, de la forma `ruta(Desde, Hasta, Longitud)`, que indica que `Desde` está unido con `Hasta` por una ruta **bidireccional** de longitud `Longitud`, donde este último es un entero nativo de Prolog.

Ejemplo:




```
mapaEjemplo([
      ruta(zarate, bsas, 30),
      ruta(bahia, bsas, 10),
      ruta(bahia, zarate, 20)]).
```

No cualquier lista de rutas es un mapa válido, y la empresa está interesada en analizar sólo aquellos mapas que lo sean. También tiene interés en estudiar diferentes caminos sobre un mapa, que considera interesantes para su operatoria.

2. Predicados pedidos

1. Definir el predicado `ciudades(+M, -Cs)`, que dado un mapa `M`, sea verdadero cuando `Cs` es una lista de todas las ciudades de `M`, sin repetidos.

   Ejemplo:

   ```
   ?- mapaEjemplo(Mapa), ciudades(Mapa, Cs)
   Cs = [zarate, bsas, bahia] ;
   No
   ```

2. Definir el predicado `ciudadesVecinas(+M, +C, -Cs)`, que dado un mapa `M` y una ciudad `C`, sea verdadero cuando `Cs` es una lista de todas las ciudades vecinas de `C`. Dos ciudades son vecinas cuando se puede llegar de una a la otra en un paso, es decir, atravesando exactamente una ruta. Tener en cuenta que las rutas son bidireccionales.

   Ejemplo:

```
?- mapaEjemplo(Mapa), ciudadesVecinas(Mapa, zarate, Cs)
Cs = [bsas, bahia] ;
No
```

3. Definir el predicado `distanciaVecinas(+M, +C1, +C2, -N)`, que dado un mapa `M` y dos ciudades **vecinas** `C1` y `C2`, sea verdadero cuando `N` es la distancia que las separa. Considerar sólo los caminos de longitud 1, o sea las rutas que las unen directamente. Observar que `C1` y `C2` pueden estar en el orden inverso al que tienen en la definición de la ruta.

   Ejemplo:

   ```
   ?- mapaEjemplo(Mapa), distanciaVecinas(Mapa, bsas, zarate, N)
   N = 30 ;
   No
   ```

4. Definir el predicado `caminoSimple(+M, +D, +H, -Cs)`, que dado un mapa `M`, un origen `D`, un destino `H` y un camino `Cs`, sea verdadero para los caminos con origen `D` y destino `H` en `M` **que no repiten ciudades**.

   Ejemplo:

   ```
   ?- mapaEjemplo(Mapa), caminoSimple(Mapa, bsas, zarate, Cs)
   Cs = [bsas, zarate] ;
   Cs = [bsas, bahia, zarate] ;
   No
   ```

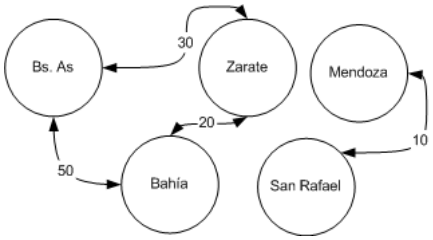
5. Definir el predicado `mapa(+M)`, donde `M` es una lista de rutas. Este predicado debe ser verdadero cuando `M` es un mapa válido. Un mapa es válido si cumple lo siguiente:

   a) Cada ciudad del mapa es alcanzable desde cualquier otra (vamos a decir que una ciudad a es alcanzable desde otra ciudad b si existe un camino, no necesariamente directo, entre a y b).

   b) No hay rutas directas desde una ciudad a si misma.

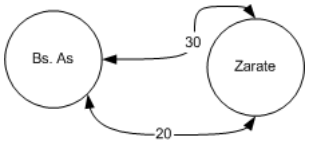
   c) No tiene ciclos triviales, es decir, no hay dos rutas directas que conecten al mismo par de ciudades (teniendo en cuenta que las rutas son caminos bidireccionales).

Ejemplos:




```
[ruta(zarate, bsas, 30),
 ruta(bahia,bsas, 50),
 ruta(bahia, zarate, 20),
 ruta(mendoza, sanrafael, 10)]
```

No es válido ya que no cumple la condición a).




```
[ruta(zarate, bsas, 30),
 ruta(bsas, zarate, 20)]
```

No es válido ya que no cumple con la condición c).

6. Definir el predicado `caminoHamiltoniano(+M, +D, +H, -Cs)` que dado un mapa `M`, un origen `D`, un destino `H` y un camino `Cs`, sea verdadero para los caminos con origen `D` y destino `H` que pasen por todas las ciudades de `M` y **no repiten ciudades**.

   Ejemplo:

```
?- mapaEjemplo(Mapa), caminoHamiltoniano(Mapa, bsas, zarate, Cs)
Cs = [bsas, bahia, zarate] ;
No
```

7. Definir el predicado `caminosHamiltonianos(+M, -Cs)` que dado un mapa `M` y un camino `Cs` que puede no estar instanciado, sea verdadero para **todos** los caminos que pasen por **todas** las ciudades del mapa (sin repetir ciudades).

   Nota: Un camino y su "capicúa" deben ser considerados como caminos distintos.

   Ejemplo:

```
?- mapaEjemplo(Mapa), caminosHamiltonianos(Mapa, Cs)
Cs = [zarate, bahia, bsas] ;
Cs = [zarate, bsas, bahia] ;
Cs = [bsas, bahia, zarate] ;
Cs = [bsas, zarate, bahia] ;
Cs = [bahia, bsas, zarate] ;
Cs = [bahia, zarate, bsas] ;
No
```

8. Definir el predicado `caminoMinimo(+M, +D, +H, -Cs, -Longitud)`, que dado un mapa `M`, un origen `D`, un destino `H` y un camino `Cs` con una longitud `Longitud` (estos dos últimos pueden no estar instanciados), sea verdadero para los caminos con origen `D` y destino `H` en `M` que no repitan ciudades y tengan longitud mínima.

   Ejemplo:

```
?- mapaEjemplo(Mapa), caminoMinimo(Mapa, bsas, zarate, Cs, N)
Cs = [bsas, bahia, zarate] N = 30 ;
Cs = [bsas, zarate] N = 30 ;
No
```

9. Definir el predicado `caminoEuleriano(+M, +D, +H, -Cs)` que dado un mapa `M`, un origen `D`, un destino `H` y un camino `Cs`, sea verdadero para los caminos con origen `D` y destino `H` que pasen por todas las rutas de `M` y **no repiten rutas**. Observar que las ciudades pueden repetirse. Notar también que un camino Euleriano no necesariamente empieza y termina en la misma ciudad, aunque así ocurra en el ejemplo.

   Ejemplo:

   ```
   ?- mapaEjemplo(Mapa), caminoEuleriano(Mapa, bsas, bsas, Cs)
   Cs = [bsas, zarate, bahia, bsas] ;
   Cs = [bsas, bahia, zarate, bsas] ;
   No
   ```

10. Definir el predicado `caminosEulerianos(+M, -Cs)` que dado un mapa `M` y un camino `Cs`, sea verdadero para todos los caminos que pasen por todas las rutas de `M` (sin repetir rutas).

    Ejemplo:

    ```
    ?- mapaEjemplo(Mapa), caminosEulerianos(Mapa, Cs)
    Cs = [zarate, bsas, bahia, zarate] ;
    Cs = [zarate, bahia, bsas, zarate] ;
    Cs = [bsas, zarate, bahia, bsas] ;
    Cs = [bsas, bahia, zarate, bsas] ;
    Cs = [bahia, bsas, zarate, bahia] ;
    Cs = [bahia, zarate, bsas, bahia] ;
    No
    ```

3. Condiciones de aprobación

El principal objetivo de este trabajo es evaluar el correcto uso del lenguaje Prolog de forma declarativa para resolver el problema planteado. Se pedirá un pequeño informe donde se explique cada predicado definido, aclarando cómo se relaciona, en caso de hacerlo, con los demás predicados (por ejemplo, indicando los predicados que hacen referencia a ellos) y cómo interviene en la solución del problema. También se debe explicitar cuáles de los argumentos de los predicados auxiliares deben estar instanciados usando `+` y `-`.

4. Pautas de entrega

Se debe entregar el código impreso con la implementación de los predicados pedidos. Cada predicado debe contar con un comentario donde se explique su funcionamiento. Asimismo, se

debe enviar un mail conteniendo el código fuente Prolog a la dirección de correo electrónico `plp-docentes@dc.uba.ar`. **Solamente** el código Prolog comentado debe acompañar el mail en forma de archivo adjunto (**no** debe incluirse el informe). El código debe poder ejecutarse en SWI-Prolog (indicar claramente cómo se debe ejecutar).

A. Algunos *predicados* y *metapredicados* definidos en SWI-Prolog

Esta sección contiene algunos predicados y metapredicados ya definidos en la actual implementación de `SWI-Prolog` que pueden ser de utilidad para el desarrollo de trabajo práctico. La descripción de cada uno se puede hallar en la ayuda de `SWI-Prolog` (invocada con el predicado `help`).

Recordar que en algunos ejercicios puede ser conveniente definir el predicado opuesto al que se pide en el enunciado, y luego usar `not`. En este caso, tener especial cuidado con la instanciación de las variables.

- Predicados sobre listas:
  - `append(-List1, -List2, -List3)`
  - `maplist(+Pred, -List)`
  - `maplist(+Pred, -List1, -List2)`
  - `maplist(+Pred, -List1, -List2, -List3)`
  - `member(-Elem, -List)`
  - `nth0(-N, -List, -Elem)`
  - `select(-Elem, -List, -Rest)`
  - `sublist(+Pred, +List1, -List2)`
  - `subset(+Subset, -Set)`
- Metapredicados:
  - `forall(+Cond, +Action)`
  - `not(+Goal)`
  - `setof(+Template, +Goal, -Set)`