Taller de Programación Lógica

Verano 2018

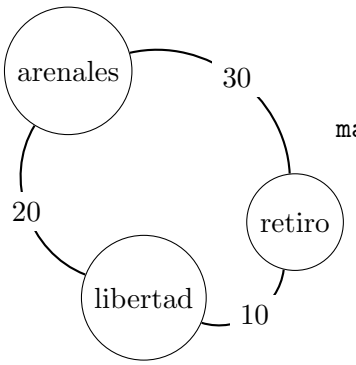
Fecha de entrega: 12 de Marzo

1. Mapas y bicisendas

El gobierno de la ciudad de *Lolipoot* desea implementar una red de ciclovías. Para ello, está diseñando un mapa con las bicisendas que pueden tomar sus ciclistas.

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

Ejemplo:




```
mapaEjemplo([
      bicisenda(arenales, retiro, 30),
      bicisenda(arenales, libertad, 20),
      bicisenda(retiro, libertad, 10)]).
```

No cualquier lista de bicisendas es un mapa válido, por lo que el gobierno de la ciudad está interesado en analizar sólo aquellos mapas que lo sean. También tiene interés en estudiar diferentes caminos sobre un mapa, que considera interesantes para sus ciclistas.

2. Predicados pedidos

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

   Ejemplo:

   ```
   ?- mapaEjemplo(Mapa), estaciones(Mapa, Es).
   Es = [arenales, retiro, libertad];
   false
   ```

2. Definir el predicado `estacionesVecinas(+M, +E, -Es)`, que dado un mapa `M` y una estación `E`, sea verdadero cuando `Es` es una lista de todas las estaciones vecinas de `E`. Dos estaciones son vecinas cuando se puede llegar de una a la otra en un paso, es decir, atravesando exactamente una bisicenda. Tener en cuenta que las bicisendas son bidireccionales.

   Ejemplo:

   ```
   ?- mapaEjemplo(Mapa), estacionesVecinas(Mapa, retiro, Es).
   Es = [libertad, arenales];
   false
   ```

3. Definir el predicado `distanciaVecinas(+M, +E1, +E2, -N)`, que dado un mapa `M` y dos estaciones **vecinas** `E1` y `E2`, sea verdadero cuando `N` es la distancia que las separa. Considerar sólo las bicisendas que las unen directamente. Observar que `E1` y `E2` pueden estar en el orden inverso al que tienen en la definición de la bicisenda.

   Ejemplo:

   ```
   ?- mapaEjemplo(Mapa), distanciaVecinas(Mapa, retiro, arenales, N).
   N = 30;
   false
   ```

4. Definir el predicado `caminoSimple(+M, +O, +D, ?C)`, que dado un mapa `M`, un origen `O`, un destino `D` y un camino `C` (que puede o no estar instanciado), sea verdadero para los caminos con origen `D` y destino `H` en `M` **que no repiten estaciones**.

   Ejemplo:

   ```
   ?- mapaEjemplo(Mapa), caminoSimple(Mapa, arenales, retiro, C).
   C = [arenales, retiro];
   C = [arenales, libertad, retiro];
   false
   ```

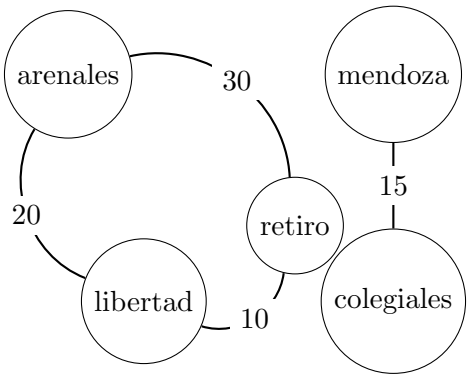
5. Definir el predicado `mapaVálido(+Bs)`, donde `Bs` es una lista de bicisendas. Este predicado debe ser verdadero cuando `Bs` es un mapa válido. Un mapa es válido si cumple lo siguiente:

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

   b) No hay bicisendas directas desde una estación a si misma.

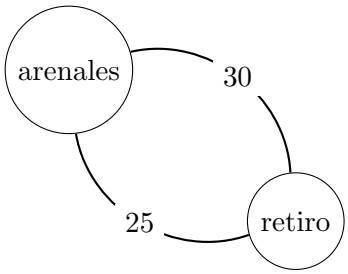
   c) No tiene ciclos triviales, es decir, no hay dos bicisendas directas que conecten al mismo par de estaciones (teniendo en cuenta que las bicisendas son caminos bidireccionales).

Ejemplos:




```
[bicisenda(arenales, retiro, 30),
 bicisenda(arenales, libertad, 20),
 bicisenda(retiro, libertad, 10),
 bicisenda(mendoza, colegiales, 15)].
```

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




```
[bicisendas(arenales, retiro, 30),
 bicisendas(retiro, arenales, 25)]
```

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

6. Definir el predicado `caminoHamiltoniano(+M, +O, +D, ?C)` que dado un mapa `M`, un origen `O`, un destino `D` y un camino `C` (que puede o no estar instanciado), sea verdadero para los caminos con origen `O` y destino `D` que pasen por todas las estaciones de `M` y **no repiten estaciones**.

   Ejemplo:

```
?- mapaEjemplo(Mapa), caminoHamiltoniano(Mapa, arenales, retiro, C)
C = [arenales, libertad, retiro];
false
```

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

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

   Ejemplo:

```
?- mapaEjemplo(Mapa), caminosHamiltonianos(Mapa, C).
C = [arenales, libertad, retiro];
C = [arenales, retiro, libertad];
C = [libertad, retiro, arenales];
C = [libertad, arenales, retiro];
C = [retiro, libertad, arenales];
C = [retiro, arenales, libertad];
false
```

8. Definir el predicado `caminoMinimo(+M, +O, +D, ?C, ?N)`, que dado un mapa `M`, un origen `O`, un destino `D` y un camino `C` con una longitud `N` (estos dos últimos pueden o no estar instanciados), sea verdadero para los caminos con origen `O` y destino `D` en `M` que no repitan estaciones y tengan longitud mínima.

   Ejemplo:

```
?- mapaEjemplo(Mapa), caminoMinimo(Mapa, arenales, retiro, C, N).
C = [arenales, libertad, retiro] N = 30;
C = [arenales, retiro] N = 30;
false
```

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 del taller. 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)`