Trabajo Práctico N°3: Programación Lógica

Paradigmas de Lenguajes de Programación
1er cuatrimestre 2020

Fecha de entrega: 14 de julio de 2020

Introducción

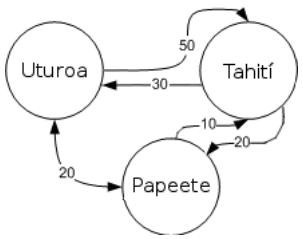
La princesa Moana[1] abrió una escuela de navegación para enseñarle a navegar a toda la gente de la isla. Una vez que la primera promoción estuvo lista para zarpar, decidieron emprender un gran desafío: explorar todas las islas de la Polinesia. Gracias a la antigua colección de mapas que encontró Moana, podían conocer la ubicación de las islas y las distancias entre ellas. El problema es que las islas son muchas y no resulta sencillo trazar rutas convenientes para poder explorarlas a todas. Es por esto que diseñó un sistema de reglas para acelerar este proceso.

Primero dibujó un mapa con todas las islas, trazando líneas entre ellas para indicar las rutas más seguras para ir desde cada una de ellas a las islas más cercanas (a las que llamaremos vecinas), y marcando en cada una de estas líneas la distancia correspondiente. Al notar que su mapa era demasiado grande para recorrerlo todo de una vez, hizo también mapas más pequeños con subconjuntos de islas para planear distintas expediciones.

Representaremos el mapa mediante una lista de adyacencia. Cada elemento de la lista es una ruta, de la forma `ruta(Desde, Hasta, Distancia)`, que indica que `Desde` está unido con `Hasta` por una ruta que recorre una distancia `Distancia`, donde este último es un entero nativo de Prolog.

Para aprovechar mejor las corrientes marinas, las rutas marcadas son unidireccionales (puede haber rutas con distinta distancia para ir y volver entre islas vecinas).

Ejemplo:




```
mapaEjemplo([
      ruta(uturoa, tahiti, 50),
      ruta(tahiti, uturoa, 30),
      ruta(papeete, uturoa, 20),
      ruta(uturoa, papeete, 20),
      ruta(tahiti, papeete, 20),
      ruta(papeete, tahiti, 10)]).
```

El objetivo es analizar distintas rutas marítimas para poder visitar las islas.

Predicados pedidos

Ejercicio 1. Definir el predicado `islas(+M, -Is)`, que dado un mapa `M`, sea verdadero cuando `Is` sea una lista de todas las islas de `M`, sin repetidas.

Ejemplo:

```
?- mapaEjemplo(Mapa), islas(Mapa, Is).
Mapa = ..., Is = [uturoa, tahiti, papeete] ;
false
```

[1] https://disney.fandom.com/es/wiki/Moana_(personaje)

Ejercicio 2. Definir el predicado `islasVecinas(+M, +I, -Is)`, que dado un mapa `M` y una isla `I`, sea verdadero cuando `Is` sea una lista de todas las islas vecinas de `I`. Dos islas son vecinas cuando se puede llegar de una a la otra en un paso, es decir, atravesando exactamente una ruta. No se debe devolver islas repetidas.

Ejemplo:

```
?- mapaEjemplo(Mapa), islasVecinas(Mapa, uturoa, Is).
Mapa = ..., Is = [tahiti, papeete] ;
false
```

Ejercicio 3. Definir el predicado `distanciaVecinas(+M, +I1, +I2, -N)`, que dado un mapa `M` y dos islas **vecinas** `I1` e `I2`, sea verdadero cuando `N` es la distancia que las separa. Considerar solamente los caminos de longitud 1, o sea las rutas que las unen directamente.

Ejemplo:

```
?- mapaEjemplo(Mapa), distanciaVecinas(Mapa, tahiti, papeete, N).
Mapa = ..., N = 20 ;
false
```

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

Ejemplo:

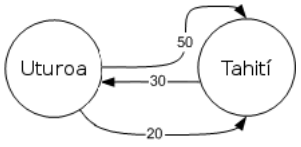
```
?- mapaEjemplo(Mapa), caminoSimple(Mapa, uturoa, papeete, C).
Mapa = ..., C = [uturoa, papeete] ;
Mapa = ..., C = [uturoa, tahiti, papeete] ;
false
```

Analizar la reversibilidad de los distintos parámetros de este predicado.

Ejercicio 5. Moana quiere asegurarse de que los mapas que dibujó sean correctos. Definir para esto 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:

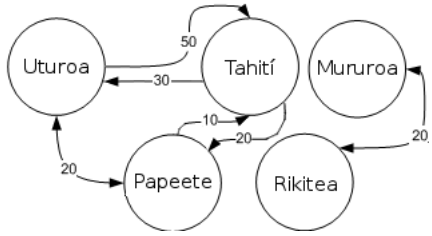
1. Cada isla del mapa es alcanzable desde cualquier otra (vamos a decir que una isla A es alcanzable desde otra isla B si existe un camino, no necesariamente directo, entre A y B).
2. No hay rutas directas desde una isla a si misma.
3. No hay dos rutas directas que conecten al mismo par de islas en el mismo sentido.

En las figuras 1 y 2 se muestran ejemplos de mapas que no son válidos.




```
[ruta(uturoa, tahiti, 50),
 ruta(tahiti, uturoa, 30),
 ruta(uturoa, tahiti, 20)]
```

Figura 1: Ejemplo de un mapa inválido (no cumple la condición 3).




```
[ruta(uturoa, tahiti, 50),
 ruta(tahiti, uturoa, 30),
 ruta(papeete, uturoa, 20),
 ruta(uturoa, papeete, 20),
 ruta(tahiti, papeete, 20),
 ruta(papeete, tahiti, 10),
 ruta(mururoa,rikitea,20),
 ruta(rikitea,mururoa,20)]
```

Figura 2: Ejemplo de un mapa inválido (no cumple la condición 1).

Ejercicio 6. Finalmente llegó el momento de trazar rutas que recorran todas las islas. Definir el predicado `caminoHamiltoniano(+M, +O, +D, -C)` que dado un mapa `M`, un origen `O`, un destino `D` y un camino `C`, sea verdadero para los caminos con origen `O` y destino `D` que pasen por todas las islas de `M` y **no repiten islas**.

Ejemplo:

```
?- mapaEjemplo(Mapa), caminoHamiltoniano(Mapa, uturoa, papeete, C).
Mapa = ..., C = [uturoa, tahiti, papeete] ;
false
```

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

Nota: caminos con las mismas islas en distinto orden deben ser considerados como caminos distintos.

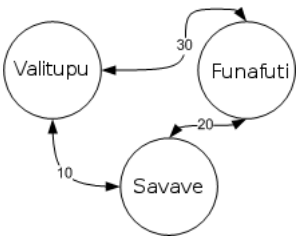
Ejemplo:

```
?- mapaEjemplo(M), caminoHamiltoniano(M, C).
M = ..., C = [uturoa, tahiti, papeete] ;    M = ..., C = [tahiti, papeete, uturoa] ;
M = ..., C = [uturoa, papeete, tahiti] ;    M = ..., C = [tahiti, uturoa, papeete] ;
M = ..., C = [papeete, tahiti, uturoa] ;    false
M = ..., C = [papeete, uturoa, tahiti] ;
```

Ejercicio 8. Definir el predicado `caminoMinimo(+M, +O, +D, -C, -Distancia)`, que dado un mapa `M`, un origen `O`, un destino `D` y un camino `C` con una distancia total `Distancia` (estos dos últimos sin instanciar), sea verdadero para los caminos con origen `O` y destino `D` en `M` que no repitan islas y cuya distancia total sea mínima.

Analizar la reversibilidad de este predicado.

Ejemplo:




```
mapaEjemplo2([
      ruta(valitupu, funafuti, 30),
      ruta(valitupu, savave, 10),
      ruta(savave, valitupu, 10),
      ruta(savave, funafuti, 20),
      ruta(funafuti, valitupu, 30),
      ruta(funafuti, savave, 20)]).
```

```
?- mapaEjemplo2(Mapa), caminoMinimo(Mapa, valitupu, funafuti, C, N).
Mapa = ..., C = [valitupu, funafuti], N = 30 ;
Mapa = ..., C = [valitupu, savave, funafuti], N = 30 ;
false
```

Pautas de Entrega

El principal objetivo de este trabajo es evaluar el correcto uso del lenguaje PROLOG de forma declarativa para resolver el problema planteado.

Se debe entregar el código impreso con la implementación de los predicados pedidos. Para cada predicado implementado (incluso los auxiliares) se debe agregar un conjunto de casos de test que muestren que el predicado exhibe la funcionalidad esperada. Para esto se debe desarrollar la sección *Tests* del archivo fuente, agregando todos los tests que consideren necesarios. Además, se debe enviar un e-mail conteniendo el código fuente en Prolog a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-PL]` seguido inmediatamente del nombre del grupo.
- El código Prolog (junto con los test desarrollados) debe acompañar el e-mail y lo debe hacer en forma de archivo adjunto con nombre `tp3.pl`.

El código debe poder ser ejecutado en SWI-Prolog. No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté adecuadamente comentado. Los objetivos a evaluar en la implementación de los predicados son:

- corrección,
- declaratividad,
- reutilización de predicados previamente definidos
- utilización de unificación, backtracking, generate and test y reversibilidad de los predicados.
- **Importante**: salvo donde se indique lo contrario, los predicados no deben instanciar soluciones repetidas ni colgarse luego de devolver la última solución. Vale aclarar que no es necesario filtrar las soluciones repetidas si la repetición proviene de las características de la entrada.

Se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

Referencias y sugerencias

Como referencia se recomienda la bibliografía incluída en el sitio de la materia (ver sección *Bibliografía* → *Programación Lógica*).

Se recomienda que utilicen los predicados ISO y los de SWI-Prolog ya disponibles, siempre que sea posible. Recomendamos especialmente examinar los predicados y metapredicados que figuran en la sección *Útil* de la página de la materia. Pueden hallar la descripción de los mismos en la ayuda de `SWI-Prolog` (a la que acceden con el predicado `help`). También se puede acceder a la documentación online de SWI-Prolog.