Paradigmas de la Programación
Obligatorio 2
Prolog Maps

Noviembre 2012

1. Introducción

Con el objeto de dar un aporte al medio ambiente y evitar los autos, se desea orientar a un ecológico peaton dentro de los límites de la ciudad de Montevideo, dándole indicaciones tanto para el uso del transporte público como también de rutas a pie. Para esto se tiene una base de datos lógica con la información geográfica de tramos de calles, números de puertas y padrones, y líneas de transporte y paradas de la ciudad de Montevideo. Esta información es de libre acceso público, y fue descargada del sitio web Sistema de Información Geográfica (http://sig.montevideo.gub.uy/) de la Intendencia Municipal de Montevideo. De forma de facilitar la tarea ya se convirtio esta información originalmente almacenada en un formato geográfico especializado a una base de datos lógica en Prolog.

Para facilitar la lectura de la información brindada al peaton se brinda ya impementada una simple interfaz gráfica basada en ventanas, la cual utiliza el servicio HTTP de Google Maps ImageAPIS (https://developers.google.com/maps/documentation/staticmaps/) para desplegar la información geográfica, como se puede apreciar en la figura 1.

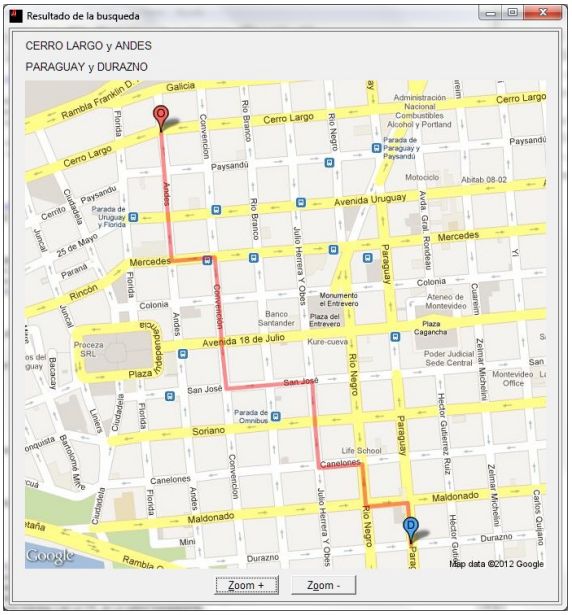

Figura 1: Interfaz gráfica - Mapa

2. Base de Datos Lógica

A continuación describimos los predicados que brindan información geográfica de Montevideo, haciendo uso de Google Maps Image para ver gráficamente la información geográfica dada por los predicados.

2.1. Tramos

El predicado `tramos/3` está definido en el archivo `tramo.pl`, y brinda información geográfica de las calles de Montevideo. Esta información viene dada por *segmentos* o de aquí en más *tramos*, los cuales tienen el nombre de la calle a la cual pertenecen y dos coordenadas en el plano: de origen y destino respectivamente. Cada coordenada a su vez está formada por dos números que representan su latitud y longitud. Por ejemplo, los siguientes dos hechos de Prolog representan dos tramos de la calle Colonia:

```
tramo('COLONIA', (-34.900698499286996, -56.17594004120682),
                 (-34.90057411351997, -56.17562724180892)).
tramo('COLONIA', (-34.90106767981142, -56.176868456186476),
                 (-34.900698499286996, -56.17594004120682)).
```

Utilizando la API de Google Maps Image se puede apreciar la ubicación de los dos tramos expuestos anteriormente. En la figura 2 podemos ver la imagen resultante, donde se muestran los segmentos en dos colores distintos. Esta imagen se obtuvo mediante la utilidad de *paths* de Google Maps Image accediendo a la URL correspondiente.

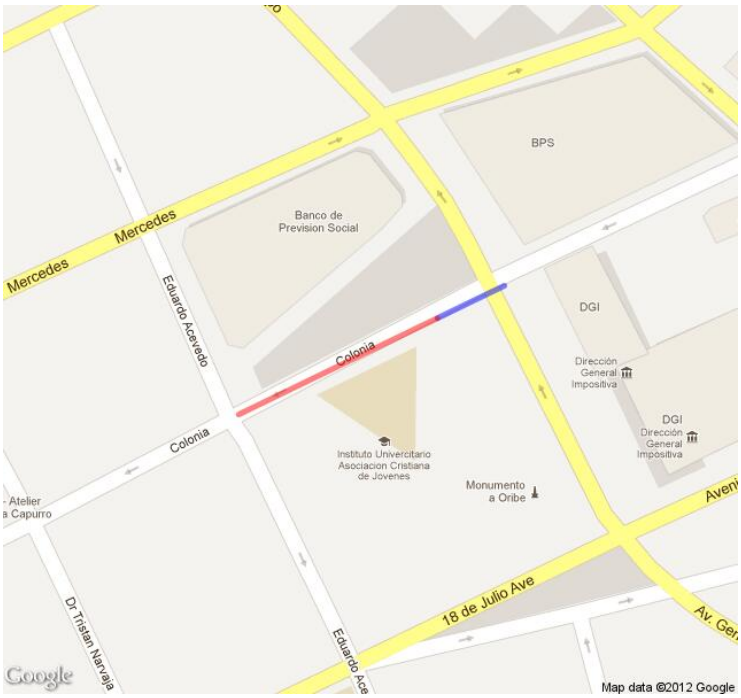

Figura 2: Dos tramos de la calle Colonia

Como se puede ver en el ejemplo anterior, una cuadra puede estar conformada por varios tramos. A su vez, los tramos no finalizan siempre en las esquinas, sino que en muchos casos comienzan o terminan adentro de una cuadra. Otra caracterísitca de los tramos es que no tienen orientación alguna, o sea un tramo cuyos extremos tienen coordenadas X e Y, puede estar almacenado de la

forma `tramo(_,X,Y)` o de la forma `tramo(_,Y,X)`. En el ejemplo anterior esto no sucede, pero de todas formas se puede verificar que la orientación de los tramos es el inverso al sentido de la calle Colonia, por tanto estos predicados no pueden usarse para encontrar un recorrido para autos.

Podemos apreciar la coordenada (−34,90057411351997, −56,17562724180892) del primer tramo se corresponde a la esquina de Fernandez Crespo y Colonia, lo comprobamos utilizando la utilidad de *markers* de Google Maps Image accediendo a la URL correspondiente, que devuelve la imagen 3.

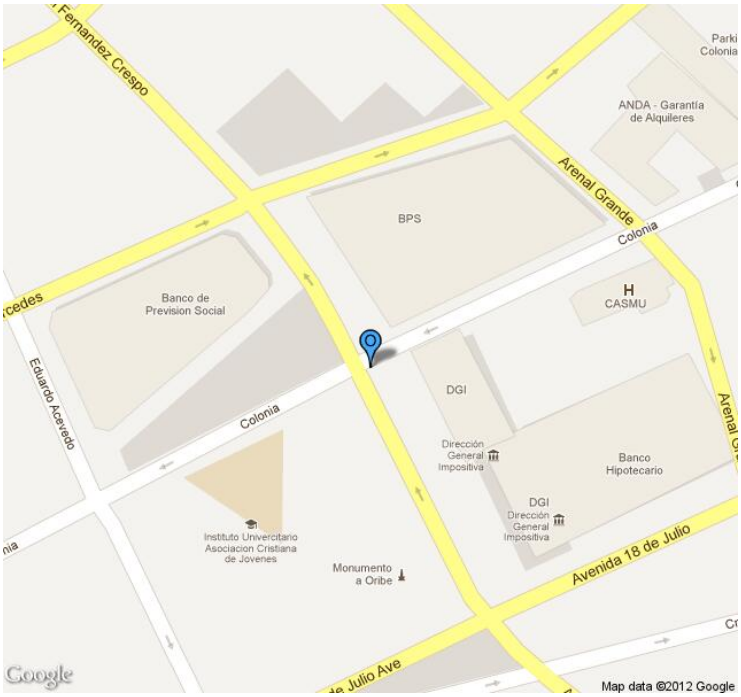

Figura 3: Esquina con coordenada (−34,90057411351997, −56,17562724180892)

También puede ocurrir que dos tramos consecutivos no coincidan en sus extremos, sino que éstos se encuentran "cercanos", debido a errores de medición en los datos. Aunque en este ejemplo los tramos confluyen exactamente a la misma coordenada, existen otras esquinas en las cuales las coordenadas no confluyen de forma exacta.
Podemos encontrar así los siguientes tramos:

```
tramo('COLONIA',(-34.90057411351997, -56.17562724180892),
                (-34.90003354253391, -56.17431164658862)).
tramo('COLONIA',(-34.900698499286996, -56.17594004120682),
                (-34.90057411351997, -56.17562724180892)).
tramo('AV DANIEL FERNANDEZ CRESPO',
                (-34.90057411351997, -56.17562724180892),
                (-34.900185456629295, -56.17593795179593)).
tramo('AV DANIEL FERNANDEZ CRESPO',
                (-34.90151938932416, -56.17504958319274),
                (-34.90057411351997, -56.17562724180892)).
```

La siguiente URL nos muestra la imagen 4 con los cuatro tramos anteriores confluyendo en la esquina.

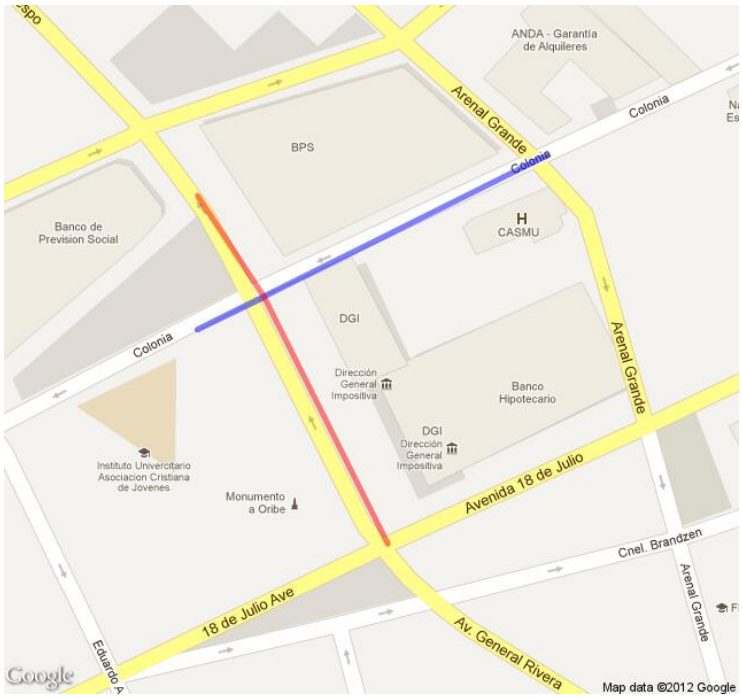

Figura 4: Tramos confluyendo en una esquina

2.2. Recorridos de Transporte Urbano

El predicado `recorrido/4` se encuentra definido en el archivo `recorrido.pl` y da la información de los recorridos del transporte urbano en Montevideo. El primer argumento de este predicado es el número de la línea de transporte, el segundo la denominación de esa línea, el tercer es un código de sublinea, y finalmente el último parámetro es una lista con las coordenadas del recorrido de la línea. Vemos a continuación un ejemplo de un hecho de Prolog para este predicado para la línea 402 que va desde la Ciudad Vieja a Malvin.

```
recorrido('402', 'CIUDAD VIEJA - MALVIN', '1',
          [ (-34.90436779781435, -56.20131082640618),
            (-34.90450384321723, -56.20163087050548), ...
```

También existe otro hecho Prolog para el recorrido de vuelta de esta misma línea el cual vemos a continuación.

```
recorrido('402', 'CIUDAD VIEJA - MALVIN', '8',
          [ (-34.89573219268848, -56.10440833150561),
            (-34.89603414794579, -56.10441304311341), ...
```

3. Descripción del Obligatorio

En este trabajo se quiere encontrar rutas tanto a pie como en ómnibus para ir de una esquina origen a otra esquina destino. Para esto se deben definir los predicados descritos en las siguientes secciones.

Módulo `list`

En el archivo `list.pl` se define este módulo, donde se deben definir los siguientes predicados:

1. `drop(+N,+Xs,?Ys)`

   Este predicado se satisface si `Ys` es la lista resultante de seguir el siguiente procedimiento: dejar un elemento de la lista `Xs` y luego saltear los `N` próximos elementos y volver a repetir este proceso hasta que no haya más elementos en `Xs`.
   Por ejemplo, `drop(2,[1,2,3,4,5,6,7,8],Xs)?` da como respuesta `Xs = [1,4,7]`. Este predicado es usado en la interfaz gráfica para reducir el detalle de recorridos muy largos debido a que Google Maps Image tiene límites en el largo de las URLs.

2. `sublist(+Xs,?Ys,+S,+E)`

   Se satisface si `Ys` es la lista formada por los elementos de la lista `Xs` que van desde la posición `S` a la posición `E`. Las posiciones se numeran desde cero, en caso que no alcancen los elementos de la lista, debe devolver los que pueda.
   Por ejemplo, `sublist([1,2,3,4],Ys,2,3)?` da como respuesta `Ys=[3,4]`. Este predicado también se usa en la interfaz gráfica para obtener parte del recorrido de un ómnibus.

Módulo `geographic`

En el archivo `geographic.pl` se deben definir los siguientes predicados geográficos.

3. `distance(+X,+Y,?D)`

   Este predicado cálcula la distancia cartesiana `D` entre las coordenadas `X` e `Y`. Estas coordenadas son pares ordenados de reales correspondientes a la latitud y longitud.

4. `inside(+X,+Y,+Z)`

   Se cumple si la coordenada `Z` está dentro del segmento definido por las coordenadas `X` e `Y` (considerar un error del $5 * 10^{-14}$ debido al redondeo).

5. `near(+X,+Y)`

   Se cumple si las coordenadas `X` e `Y` se consideran "cercanas", esto es, si están a una distancia menor o igual a $5 * 10^{-14}$.

6. `cross(+X1,+X2,+Y1,+Y2,?Z)`

   Se cumple si los segmentos determinados por las coordenadas `X1,X2` e `Y1,Y2` se intersectan en la coordenada `Z`.

7. `corner(+C1,+C2,?X)`

   Se cumple si X es una coordenada "cercana" a la esquina formada por las calles `C1` y `C2`. Esta condición se cumple si se satisface alguna de las siguientes condiciones:

- existen dos tramos, correspondientes a las calles `C1` y `C2` respectivamente, que se intersectan en `X`
- exsiten dos extremos de tramos "cercanos", correspondientes a las calles `C1` y `C2` respectivamente

Estas condiciones se deberán evaluar en el orden dado, y si la primera condición se satisface se deberá cortar los siguientes resultados posibles. Por ejemplo, la consulta `corner('COLONIA','AV DANIEL FERNANDEZ CRESPO',X)` debe satisfacerse para la sustitución `X = (-34.90057411351997, -56.17562724180892)`, tal cual puede verificarse en la imagen 3.

Módulo `pathplanning`

Se pide definir los siguientes predicados en el archivo `pathplanning.pl` para la resolución de recorridos a pie:

8. `adyacentes(+X,?Y)`

   Dada una coordenada `X`, este predicado se cumple si se satisface alguna de las siguientes condiciones:

   - existe una coordenada `Z` que está cerca de `X`, y hay un tramo de `Y` a `Z` o de `Z` a `Y`,
   - `X` está adentro de un tramo de `Y` a `Z` o de `Z` a `Y`, y `X` no está cerca de `Y` ni de `Z`.

9. `caminoHC(+C1,+C2,+C3,+C4,?P)`

   Se cumple si `P` es una lista de coordenadas con un camino desde la esquina formada por las calles `C1` y `C2`, y la esquina formada por `C3` y `C4`. Esta búsqueda deberá realizarse utilizando el método de "Hill Climbing" visto en el teórico.

10. `caminoAmp(+C1,+C2,+C3,+C4,?P)`

    Predicado equivalente al anterior pero haciendo una búsqueda en amplitud en el espacio de búsqueda.

11. `caminoAast(+C1,+C2,+C3,+C4,?P)`

    Predicado equivalente al anterior pero donde la búsqueda se realiza utilizando el algoritmo A*. Para la estimación heurística se deberá usar la distancia en línea recta al destino.

Módulo `buses`

Los predicados de este módulo permitirán encontrar el mejor ómnibus para ir desde un origen a un destino.

12. `findBusCoor(+O,+D,?N,?V,?Desc,?PO,?PD)`

    Se satisface si el ómnibus número `N` con código de sublinea `V` y descripción `Desc` tiene el recorrido que minimiza la suma de las distancias formadas por:

    - la coordenada origen `O` y la coordenada más cercana a ésta en la lista de coordenadas que conforman el recorrido del ómnibus
    - la coordenada destino `D` y la coordenada más cercana a ésta en el recorrido del ómnibus

Además `PO` es el índice de la coordenada en el recorrido más cercana a `O`, y `PD` es el índice en el recorrido pero de la coordenada más cercana a `D`. Dado que se desea ir desde `O` hasta `D` se tiene que cumplir que la coordenada más cercana al origen `O` tiene que ocurrir antes en la lista que conforma el recorrido que la coordenada más cercana al destino `D` (o sea, se debe cumplir `PO < PD`).

13. `findBusCalles(Co1,Co2,Cd1,Cd2,N,V,Desc,PO,PD)`

    Predicado análogo al anterior pero en vez de utilizar las coordenadas origen y destino, utiliza los nombres de las calles que conforman la esquina origen `Co1` y `Co2`, y la esquina destino `Cd1` y `Cd2`.

Módulo `proxy`

En este módulo se define el predicado:

```
proxy(-P).
```

En caso de necesitar un proxy *http* para el acceso a internet se debe configurar mediante la instanciasión correcta de este predicado. En caso de tener acceso directo a internet instanciar con la lista vacía `P = [ ]`, por ejemplo, si se usa una laptop conectada a la Wi-Fi de ORT se está en este caso. En caso de necesitar configurar un proxy *http* para el acceso a internet se debe instanciar con `P = [proxy(IP_PROXY,PUERTO_PROXY)]`, donde `IP_PROXY` es la IP del proxy, y `PUERTO_PROXY` es el número de puerto. Por ejemplo, para las máqunias de laboratorio de ORT se debe instanciar de la siguiente forma `P = [proxy('192.168.57.2',80)]`.

Módulo Principal `ventanas`

Finalmente en el archivo `ventanas.pl` se define el predicado `display/0`, que es el encargado de levantar la interfaz gráfica. En este archivo se utilizan todas las funciones definidas en los módulos descriptos anteriormente, por lo que **es importante utilizar los nombres y respetar las especificaciones de los predicados tal cual son descritos en las secciones anteriores**.

La interfaz gráfica es muy sencilla de utilizar, y pide ingresar los datos de origen y destino, así como la búsqueda que se quiere realizar. En la figura 5 se muestra un ejemplo de uso. Tener en cuenta que los nombres de las calles deben ir en mayúscula, y deben ser exactamente los mismos que figuran en la información de tramos:

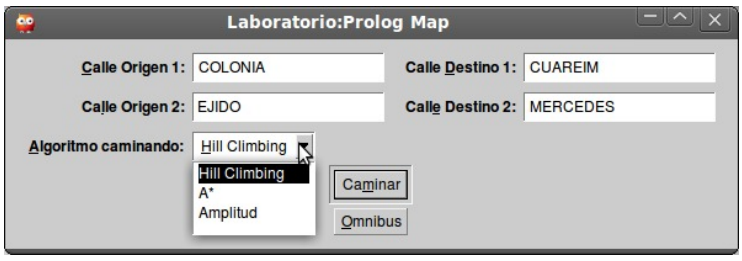

Figura 5: Interfaz gráfica - Consultas

Este módulo va almacenando las imagenes obtenidas de Google Maps Image en el directorio donde se ejecuta la aplicación, estas imagenes se van nombrando numéricamente de forma consecutiva ("0.jpg", "1.jpg" y así sucesivamente). Las imagenes no deben ser borradas mientras la consola Prolog esté corriendo, esto puede producir efectos indeseados como no obtener la última imágen

pedida sino una cacheada en una consulta previa anteriormente. Luego de terminada la sesión Prolog sí pueden ser borradas sin inconveniente alguno.

Dentro de este módulo también se utiliza el módulo `staticmaps`, el cual también proveemos, este brinda facilidades para el manejo de las URLs de Google Maps Image.

Resumiendo: para poder levantar la interfaz gráfica se necesitan implementar los predicados pedidos en los módulos

- `list`
- `geographic`
- `pathplanning`
- `buses`

Por otra parte también son necesarios los siguientes módulos que ya se brindan implementados y no se pueden modificar:

- `ventanas`
- `staticmaps`
- `proxy`
- `tramo` y `recorrido` que almacenan los hechos de la base de datos lógica.

4. Entregables

Se deben entregar los archivos `.pl` listos para ser compilados en SWI Prolog. No se piden reportes extra, pero los programas deben ser **legibles** y estar **comentados** apropiadamente, con la descripción de la implementación de los predicados pedidos y de los predicados auxiliares utilizados. Se tendrá en cuenta el **buen dieseño** de la solución y la **legibilidad** de los programas. Se debe entregar todo el código en forma impresa y en un CD, en un sobre transparente. La entrega se realizará en Bedelía con boleta de entrega de obligatorio.

5. Fechas

- Lectura: 5/11/12
- Puntaje (máx): 17 puntos
- Fecha de Entrega: 5/12/12
- Defensa: A fijar con los docentes