Trabajo Práctico N°3: Programación Lógica

Paradigmas de Lenguajes de Programación
1er cuatrimestre 2023

Fecha de entrega: 29 de junio

Introducción

Este trabajo práctico consiste en definir hechos y reglas que nos permitan modelar y recorrer laberintos. Los laberintos solo pueden contener tierra, paredes, hielo y tesoros.

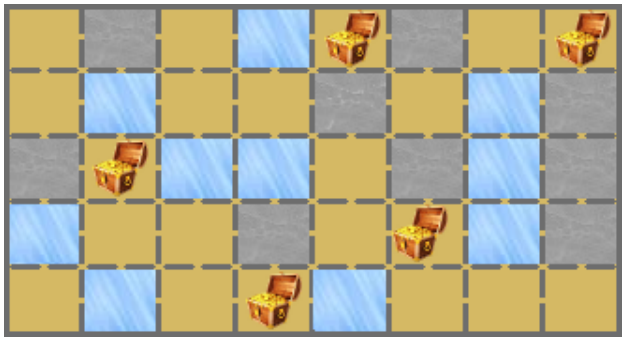

Figura 1: un laberinto de ejemplo.

Un laberinto se representará con una lista de filas. Cada fila será una lista de celdas, cada cual con su correspondiente contenido, el cual será *tierra*, *hielo*, *pared* o *tesoro*. Cada posición del laberinto se representará con una tupla de la forma $(Fila, Columna)$, ambos indexados desde 1, siendo $(1, 1)$ la celda superior izquierda.

El jugador ingresa al laberinto por la izquierda y lo recorre intentando encontrar la salida (a la derecha), recogiendo tesoros en el camino. En otras palabras, todas las celdas de la columna 1 (a la izquierda) se consideran entradas (a menos que sean paredes) y todas las celdas de la última columna se consideran salidas.

El jugador que recorrerá el laberinto no puede atravesar paredes ni tampoco ir más allá de los límites del laberinto, y solo puede moverse en las siguientes direcciones: *arriba*, *abajo*, *derecha*, *izquierda*.

Durante su recorrido, el jugador podrá pasar por celdas de salida, sin necesidad de salir por ellas.

Al pisar una celda cubierta de hielo, el jugador patinará hacia la celda siguiente en la misma dirección en la que venía avanzando, a menos que esta sea una pared o uno de los límites del laberinto, en cuyo caso permanecerá en dicha celda. Si la celda siguiente también tiene hielo, seguirá patinando en la misma dirección hasta llegar a una celda sin hielo o chocar contra una pared o con el borde del laberinto.

Se puede comenzar el camino por cualquier celda de la columna 1 (la izquierda) que no sea una pared, y se sale por cualquier celda de la derecha en la que sea posible pararse (es decir, que sea tierra, tesoro, o una celda con hielo a la que se llegó de manera tal que no se pueda seguir patinando).

Se considera que el primer paso (para ingresar al laberinto) se da siempre hacia la derecha, por lo cual si la primera celda que se pisa contiene hielo el jugador patinará en esa dirección.

Ejercicios

En los siguientes ejercicios podrán usar ! (cut) y predicados de alto orden cuando sea necesario. Se recomienda repasar la sección "Útil" de la página de la materia, en particular la parte de "metapredicados".

Ejercicio 1. Definir el predicado `siguiente(+X, +Y, ?Dir, ?X2, ?Y2)` que define la relación entre una posición, una dirección y la posición de la coordenada siguiente en dicha dirección.

Ejercicio 2. Definir el predicado `dimensiones(+Laberinto, ?Columnas, ?Filas)` que es verdadero si el laberinto tiene dimensión $Columnas \times Filas$. Por ejemplo:

```
?- dimensiones([[tierra,hielo,tierra], [pared, tierra, tesoro]], C, F).
C = 3,
F = 2.
```

No es necesario verificar que el laberinto sea válido, puede suponerse que existe al menos una fila y que todas tienen la misma longitud.

Ejercicio 3. Definir el predicado `esLaberinto(+L)` que es verdadero cuando el laberinto es válido. En el contexto de este trabajo un laberinto válido es aquel cuyas filas son todas de la misma longitud, que contiene al menos una celda, y en el que todas las celdas tienen un contenido válido.

Ejercicio 4. Definir el predicado `celda((+X, +Y), +Laberinto, ?Contenido)` que relaciona una posición en un laberinto con su contenido. La variable `X` se corresponde con la fila, e `Y` con la columna.

Ejercicio 5. Definir el predicado `tesoros(+Laberinto, -Tesoros)` que relaciona a un laberinto con la lista de coordenadas de las celdas cuyo contenido es un tesoro (sin repeticiones). Basta con generar una única solución (no importa el orden).

Sugerencia: usar `findall` o `setof`.

Ejercicio 6. Definir el predicado `paso(+CoordInicial,+Dir,+Laberinto,-Recorrido)` que dados una coordenada, una dirección y un laberinto instancia la lista de posiciones que se recorren como resultado de avanzar un paso en esa dirección (la lista tendrá longitud 1 a menos que el jugador patine, en cuyo caso la longitud será mayor, o que no pueda avanzar en dicha dirección, en cuyo caso la lista estará vacía). El recorrido no incluye la posición donde se encuentra el jugador antes de dar el paso.

Sugerencias: escribir un predicado auxiliar que instancie la siguiente posición y su contenido dadas una posición y una dirección, y otro que determine si el jugador puede detenerse al caer en una celda con un contenido determinado.

Indicar en un comentario si este predicado es o no **reversible** en cada uno de sus parámetros. Justificar. (Si lo es, pueden usarlo con la instanciación más general de aquí en adelante).

Ejercicio 7. Definir el predicado:
`caminoDesde(+Laberinto, +CoordInicial, ?Dir, -Camino, +Longitud)` que genera todos los recorridos de la longitud dada (medida en cantidad de **pasos**) que se pueden realizar en ese laberinto comenzando en la posición dada y dando el primer paso en la dirección (que puede venir instanciada o instanciarse).

Ejercicio 8. Definir el predicado `solucion(+Laberinto, -Camino)` que, dado un laberinto, genera todos los caminos posibles que comienzan en alguna entrada del laberinto y que llegan a una salida. Además, se agrega la restricción de que la longitud del camino es a lo sumo tan larga como el laberinto (si bien se pueden repetir celdas, la solución no debe ser más larga que la que pasa por todas las celdas).

Nota: la celda de entrada debe formar parte de la solución. Una opción para facilitar esto es comenzar un camino desde la columna 0, fuera del laberinto, y empezar avanzando hacia la derecha; alternativamente se puede empezar en la fila 1, verificando que la celda inicial no sea una pared, y agregar dicha celda a la solución.

Ejercicio 9. Definir el predicado `solucionOptima(+Laberinto, -Camino)` que dado un laberinto produce las soluciones más cortas que pasan por todos los tesoros.

1. Pautas de Entrega

Cada predicado debe contar con un comentario donde se explique su funcionamiento. Cada predicado asociado a los ejercicios debe contar con ejemplos (tests) que muestren que exhibe la funcionalidad solicitada. 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 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
- Uso de unificación, backtracking, generate and test y reversibilidad de los predicados que correspondan.
- **Importante**: salvo donde se indique lo contrario, los predicados no deben instanciar soluciones repetidas. Vale aclarar que no es necesario filtrar las soluciones repetidas si la repetición proviene de las características de la entrada.

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

2. Referencias y sugerencias

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

Se recomienda que, siempre que sea posible, utilicen los predicados ISO y los de SWI-Prolog ya disponibles. Recomendamos especialmente examinar los predicados y metapredicados que figuran en la sección *Otros* 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.