Trabajo Práctico N°3: Programación Lógica

Paradigmas de Lenguajes de Programación
2do cuatrimestre 2023

Fecha de entrega: 16 de noviembre

Introducción

Este trabajo consiste en programar en Prolog un conjunto de predicados para resolver coloreo de grafos (no dirigidos). Además del coloreo tradicional se agregan algunas restricciones y variaciones a lo largo de los ejercicios.

Dado un grafo $G = (V, E)$ no dirigido, un coloreo Γ es una asignación de colores a nodos de V tal que:

$$\forall (v, w) \in E \cdot \Gamma(v) \neq \Gamma(w)$$

En el contexto del trabajo un color va a estar representado por un número positivo. De esta manera una paleta P indica que se pueden usar solamente los colores $1 \dots P$.

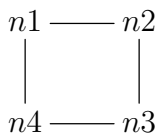
1. Grafos

Los grafos van a ser representados por la lista de aristas E. Notar que de esta manera quedan descartados aquellos grafos que tienen nodos sin vecinos. Se asume que E no tiene elementos repetidos, ni elementos simétricos. O sea, si $(v, w) \in E$ entonces $(w, v) \notin E$.

Se cuenta con el predicado `ejemplo(+Codigo, -E)` en el cual se definen grafos de ejemplo a ser utilizados más adelante.

```
ejemplo(c4, [(n1,n2),(n2,n3),(n3,n4),(n4,n1)]).
```

El grafo representado por `E` en `ejemplo(c4,E)` es:



Para representar una instancia coloreada del grafo anterior donde los nodos `n1` y `n3` tienen el color `1` y los restantes el color `2`, se utilizará la siguiente estructura:

```
grafo(
[(n1,1),(n2,2),(n3,1),(n4,2)],    % coloreo
[(n1,n2),(n2,n3),(n3,n4),(n4,n1)]) % lista de aristas
```

La primera componente deberá tener un elemento por cada nodo en el grafo, sin orden particular.

Ejercicio 1.

Definir `armar_grafo(+E,-Grafo)` que dada una lista de aristas, deje en `Grafo` una instancia de la estructura antes dicha correspondiente a `E`, donde los colores asignados sean variables distintas.

```
?- ejemplo(c4,E), armar_grafo(E, G).
G = grafo([ (n2, _G410), (n3, _G416), (n4, _G422), (n1, _G428)],
          [ (n1, n2), (n2, n3), (n3, n4), (n4, n1)]) ;
false.
```

2. Coloreo

Ejercicio 2. Definir el predicado `color_nodo(+Grafo, +Nodo, ?Color)` que es verdadero si en el grafo, el nodo de nombre indicado tiene dicho color.

```
?- ejemplo(c4,E), armar_grafo(E, G), color_nodo(G, n2, X).
G = grafo([ (n2, X), (n3, _G477), (n4, _G483), (n1, _G489)],
          [ (n1, n2), (n2, n3), (n3, n4), (n4, n1)]) ;
false.
```

Ejercicio 3. Definir el predicado `vecino(+Grafo, ?V, ?W)` que indica si `V` y `W` son vecinos en el grafo.

```
?- ejemplo(c4,E), armar_grafo(E, G), vecino(G, n1, X).
X = n2 ;
X = n4 ;
false.
```

Ejercicio 4. Definir el predicado `colores_vecinos(+Grafo, +Nodo, -Colores)` que calcula los colores *ya fijados* en el grafo para los vecinos del nodo. Si el color no está instanciado, deberá ignorarse.

```
?- ejemplo(c4, E), armar_grafo(E, G), color_nodo(G, n4, 3),
  colores_vecinos(G, n1, LC).
LC = [3] ;
false.
```

Ejercicio 5. Definir `pintar_nodo(+Paleta, +?Grafo, +Nodo)` para que, dada una paleta, pinte el nodo indicado en el grafo de todas las maneras posibles. Se debe asegurar que el color asignado para el nodo no sea uno compartido por sus vecinos.

```
?- ejemplo(c4, E), armar_grafo(E, G), color_nodo(G, n4, 3), pintar_nodo(4, G, n1).
G = grafo([ (n2, _G508), (n3, _G514), (n4, 3), (n1, 1)],
  [ (n1, n2), (n2, n3), (n3, n4), (n4, n1)]) ;
G = grafo([ (n2, _G508), (n3, _G514), (n4, 3), (n1, 2)],
  [ (n1, n2), (n2, n3), (n3, n4), (n4, n1)]) ;
G = grafo([ (n2, _G508), (n3, _G514), (n4, 3), (n1, 4)],
  [ (n1, n2), (n2, n3), (n3, n4), (n4, n1)]) ;
false.
```

Ejercicio 6. Definir `pintar_grafo(+Paleta,+?Grafo)` de modo que pinte todos los nodos del grafo con las mismas consideraciones que el ejercicio anterior. Además, para los nodos que ya están pintados, se debe demorar lo menos posible y saltearlos.

```
?- ejemplo(c4, E), armar_grafo(E, G), pintar_grafo(2, G).
G = grafo([ (n2, 1), (n3, 2), (n4, 1), (n1, 2)],
  [ (n1, n2), (n2, n3), (n3, n4), (n4, n1)]) ;
G = grafo([ (n2, 2), (n3, 1), (n4, 2), (n1, 1)],
  [ (n1, n2), (n2, n3), (n3, n4), (n4, n1)]) ;
false.

?- ejemplo(c4, E), armar_grafo(E, G), pintar_grafo(1, G).
false.
```

Ejercicio 7. Definir `mismo_color(+?Grafo, +V, +W)`, que establezca que, en el grafo, ambos nodos tendrán el mismo color.

```
?- ejemplo(c4, E), armar_grafo(E, G), mismo_color(G, n1, n3), pintar_grafo(3, G).
G = grafo([ (n2, 1), (n3, 2), (n4, 1), (n1, 2)],
  [ (n1, n2), (n2, n3), (n3, n4), (n4, n1)]) ;
G = grafo([ (n2, 1), (n3, 2), (n4, 3), (n1, 2)],
  [ (n1, n2), (n2, n3), (n3, n4), (n4, n1)]) ;
G = grafo([ (n2, 1), (n3, 3), (n4, 1), (n1, 3)],
  [ (n1, n2), (n2, n3), (n3, n4), (n4, n1)]) ;
G = grafo([ (n2, 1), (n3, 3), (n4, 2), (n1, 3)],
  [ (n1, n2), (n2, n3), (n3, n4), (n4, n1)]) ;
... 12 resultados en total
```

Ejercicio 8. Definir `es_valido(+Grafo)` para que sea verdadero solo si el coloreo del grafo es válido. Se asume que todos los nodos del grafo tienen un color asignado. Recordar que un coloreo es válido cuando dos vecinos no tienen asignado el mismo color.

Ejercicio 9. Definir `coloreo(+Grafo, -Coloreo)` para que realice todos los coloreos válidos usando a lo sumo tantos colores como nodos hay en el grafo. En `Coloreo` debe quedar cada uno de los coloreos resultantes.

Además de ser coloreos válidos, estos no deben dejar huecos. O sea, si el color c está en el grafo, entonces los colores $1 \dots c$ también.

```
?- ejemplo(c4, E), armar_grafo(E, G), coloreo(G, C).
C = [ (n2, 1), (n3, 2), (n4, 3), (n1, 4)] ;
C = [ (n2, 1), (n3, 2), (n4, 3), (n1, 2)] ;
... 38 resultados en total
```

3. Pautas de Entrega

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.

4. 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.