

## Introducción a la Inteligencia Artificial

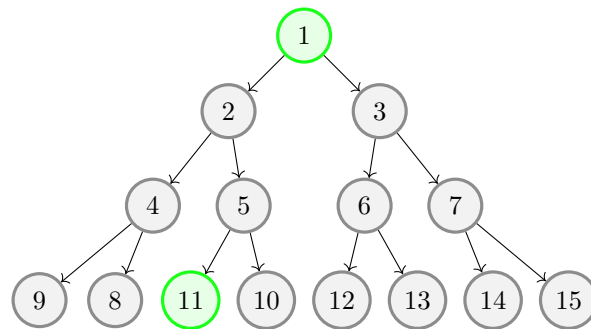
**Actividad 2**

Leonardo Flores Torres

19 de diciembre de 2022

Elabore un programa en `prolog` con el espacio de estados que comienza con el número 1 y la función sucesor para el estado  $n$  devuelve 2 estados, los números  $2n$  y  $2n + 1$ .

- Dibuje la porción de estados para los estados del 1 al 15. Supongamos que el estado meta es el 11.
- Programe los predicados con las reglas necesarias para encontrar el orden en que serán visitados los nodos en profundidad y amplitud.

**Solución:**

Para resolver el caso de búsqueda en profundidad primero se declararon las conexiones entre nodos como se muestra a continuación:

```
1  % Conexiones entre nodos
2  conexion(inicio,1).
3  conexion(1,2).
4  conexion(1,3).
5  conexion(2,4).
6  conexion(2,5).
7  conexion(3,6).
8  conexion(3,7).
9  conexion(4,8).
10 conexion(4,9).
11 conexion(5,10).
12 conexion(5,11).
13 conexion(6,12).
14 conexion(6,13).
```

```

15 conexion(7,14).
16 conexion(7,15).
17 conexion(11,fin).

```

No es necesario declarar la regla `conexion(inicio,1)` ni `conexion(11,fin)`, se podría cambiar las ocurrencias de `1` por `inicio` y las de `11` por `fin`,

- `conexion(inicio,2)`,
- `conexion(inicio,3)`,
- `conexion(5,fin)`.

Lo que se logra al haber escrito las conexiones como se hizo es el definir una conexión auxiliar que apunta a los nodos de inicio y de término, `1` y `11`, respectivamente. Posteriormente se define la meta para indicar cuando se ha alcanzado nuestro nodo objetivo:

```

18 %% Declaracion de la meta
19 meta(fin).

```

Con la meta y las conexiones declaradas es necesario definir las reglas que indiquen las relaciones entre nodos, esto es, permitir a `prolog` identificar como un nodo está conectado con otro:

```

20 %% Reglas para definir las condiciones de como dos nodos estan conectados
21 sucesor(Posicion1, Posicion2) :- conexion(Posicion1, Posicion2).
22 sucesor(Posicion1, Posicion2) :- conexion(Posicion2, Posicion1).

```

Finalmente se agrega el algoritmo de búsqueda en profundidad:

```

23 %% Algoritmo de busqueda en profundidad
24 ruta([fin|RestoDeRuta], [fin|RestoDeRuta]).
25 ruta([PosicionActual|RestoDeRuta], Sol) :-
26     sucesor(PosicionActual, PosicionSiguiente),
27     not(member(PosicionSiguiente, RestoDeRuta)),
28     ruta([PosicionSiguiente,PosicionActual|RestoDeRuta], Sol).

```

Si se permite a `prolog` mostrar los pasos que realiza para ejecutar el algoritmo se podrá observar que efectivamente, la búsqueda se efectúa en profundidad, esto se activa con el comando `trace.`, y `notrace.` si no se quiere mostrar ya los pasos de ejecución.

Para requerir la solución de la búsqueda se escribió la siguiente regla:

```

29 %% Solicitar solucion
30 solucion(Inicio, Sol) :-
31     ruta([Inicio], SolAux),
32     reverse(SolAux, Sol).

```

Al ejecutar `solucion` con el punto de partida al punto de llegada deseados en `prolog`, en la terminal, se obtiene como resultado lo siguiente:

```

1  ?- [searchDepth].
2  true.

3  ?- solucion(inicio,S).
4  S = [inicio, 1, 2, 5, 11, fin] ;
5  false.
```

Ahora, para el caso de búsqueda en amplitud primero se declaran las conexiones que existen como en el caso anterior:

```

1  % Conexiones entre nodos
2  conexion(1,2).
3  conexion(1,3).
4  conexion(2,4).
5  conexion(2,5).
6  conexion(3,6).
7  conexion(3,7).
8  conexion(4,8).
9  conexion(4,9).
10 conexion(5,10).
11 conexion(5,11).
12 conexion(6,12).
13 conexion(6,13).
14 conexion(7,14).
15 conexion(7,15).
```

Las conexiones en realidad son las mismas, la única diferencia que se hizo aquí fue remover `inicio` y `fin` para demostrar que se puede hacer la búsqueda sin esas conexiones auxiliares. La meta sigue siendo la misma, solamente se sustituye `fin` por `11`:

```

16 %% Declaracion de la meta
17 meta(11).
```

Ahora, el algoritmo para la búsqueda en amplitud es el siguiente:

```

18 %%% La ruta en realidad es una lista de caminos que se van creando
19 %%% conforme se visitan los nodos en amplitud
20 ruta([[NodoActual|Camino]|_], [NodoActual|Camino]) :-
21     meta(NodoActual).
22 ruta([Camino|Caminos], Sol) :-
23     extender(Camino, NuevosCaminos),
24     append(Caminos, NuevosCaminos, Caminos1),
25     ruta(Caminos1, Sol).
26
27 extender([NodoActual|Camino], NuevosCaminos) :-
28     bagof([NuevoNodo,NodoActual|Camino],
29         (conexion(NodoActual, NuevoNodo),
```

```
30     not((member(NuevoNodo, [NodoActual|Camino]))),  
31     NuevosCaminos),  
32     !.  
33 %% Caso si ya no hay sucesores en Camino  
34 extender(-, []).
```

De manera similar al caso anterior se define una regla `solucion` para encontrar la solución de la búsqueda:

```
35 %% Búsqueda primero en Amplitud  
36 solucion(Inicio, Sol) :-  
37     ruta([[Inicio]], SolAux),  
38     reverse(SolAux, Sol).
```

Y realizando la búsqueda en terminal se obtiene el siguiente resultado:

```
1  ?- [searchAmp].  
2  true.  
3  
4  ?- solucion(1,S).  
5  S = [1, 2, 5, 11] ;  
6  false.
```

---

## Referencias

- [1] Ivan Bratko. *Prolog programming for artificial intelligence*. Pearson education, 2012.
- [2] Leon Sterling and Ehud Y Shapiro. *The art of Prolog: advanced programming techniques*. MIT press, 1994.
- [3] Marcus Triska. The power of prolog. <https://www.metalevel.at/prolog>, 2005. Visitado: 2022-09-11.
- [4] Alejandro Guerra-Hernandez. Programación para la inteligencia artificial. <https://www.uv.mx/personal/aguerra/pia/>, 2022. Visitado: 2022-10-07.