Universidad Veracruzana

Inteligencia Artificial

Introducción a la Inteligencia Artificial

Actividad 2

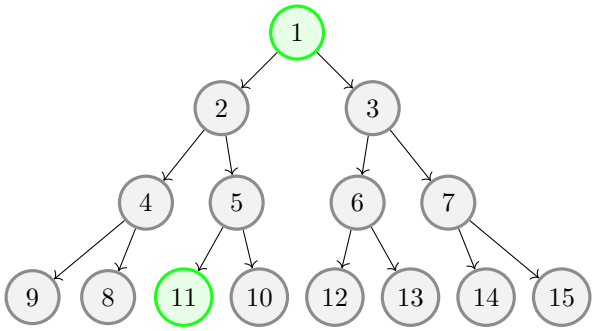
Leonardo Flores Torres

20 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:

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

```
conexion(7,14).
conexion(7,15).
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:

```
%%% Declaracion de la meta
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:

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

Finalmente se agrega el algoritmo de búsqueda en profundidad:

```
%%% Algoritmo de busqueda en profundidad
ruta([fin|RestoDeRuta], [fin|RestoDeRuta]).
ruta([PosicionActual|RestoDeRuta], Sol) :-
    sucesor(PosicionActual, PosicionSiguiente),
    not(member(PosicionSiguiente, RestoDeRuta)),
    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 efectua 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:

```
%%% Solicitar solucion
solucion(Inicio, Sol) :-
    ruta([Inicio], SolAux),
    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:

```
?- [searchDepth].
true.

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

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

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

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

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

```
%%% La ruta en realidad es una lista de caminos que se van creando
%%% conforme se visitan los nodos en amplitud
ruta([[NodoActual|Camino]|_], [NodoActual|Camino]) :-
    meta(NodoActual).
ruta([Camino|Caminos], Sol) :-
    extender(Camino, NuevosCaminos),
    append(Caminos, NuevosCaminos, Caminos1),
    ruta(Caminos1, Sol).

extender([NodoActual|Camino], NuevosCaminos) :-
    bagof([NuevoNodo,NodoActual|Camino],
        (conexion(NodoActual, NuevoNodo),
```

```
        not((member(NuevoNodo, [NodoActual|Camino]))),
        NuevosCaminos),
        !.
%%% Caso si ya no hay sucesores en Camino
extender(_, []).
```

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

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

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

```
?- [searchAmp].
true.

?- solucion(1,S).
S = [1, 2, 5, 11] ;
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.