LABORATORIOS DE INTELIGENCIA ARTIFICIAL
CURSO 2018–2019

PRÁCTICA 3: PROLOG

Fecha de publicación: 2019/03/27

Fechas de entrega: grupos jueves: miércoles 2019/04/10, 23:55
grupos viernes: jueves 2019/04/11, 23:55

Planificación: Ejercicios para la 1.ª semana: 1, 2, 3 y 4.
Ejercicios para la 2.ª semana: 5, 6, 7 y 8.

Entrega: Para la entrega en formato electrónico, se creará un archivo zip que contenga todo el material de la entrega, cuyo nombre, todo él en minúsculas y sin acentos, tildes, o caracteres especiales, tendrá la siguiente estructura:

[gggg]_p3_[mm]_[apellido1]_[apellido2].zip

donde

[gggg] : Número de grupo: (2301, 2311, 2312, etc.)
[mm] : Número de orden de la pareja con dos dígitos (01, 02, 03,...)
[apellido1] : Primer apellido del miembro 1 de la pareja
[apellido2] : Primer apellido del miembro 2 de la pareja

Los miembros de la pareja aparecen en orden alfabético. Ejemplos :

- 2311_p3_01_delval_sanchezmontanyes.zip (práct. 3 de la pareja 01 en el grupo de prácticas 2311)
- 2362_p3_18_salabert_suarez.zip (práctica 3 de la pareja 18 en el grupo de prácticas 2362)

Contenido del fichero comprimido:

- El archivo de código Prolog: [gggg]_p3_[mm].pl
- El archivo: [gggg]_p3_[mm]_readme.txt [OPCIONAL]
- El archivo de la memoria (nombre siempre en minúsculas): [gggg]_p3_[mm]_memoria.pdf

---

EJERCICIO 1 (1 punto). Implemente un predicado duplica(L,L1), que es cierto si la lista L1 contiene los elementos de L duplicados

```
?- duplica([1, 2, 3], [1, 1, 2, 2, 3, 3]).
true.

?- duplica([1, 2, 3], [1, 1, 2, 3, 3]).
false.

?- duplica([1, 2, 3], L1).
L1 = [1, 1, 2, 2, 3, 3].

?- duplica(L, [1, 2, 3]).
false.
```

Utiliza los predicados trace y notrace para trazar la ejecución de los objetivos anteriores, consulta el tutorial de Prolog para ver cómo se usan.

EJERCICIO 2 (1 punto). Implementa el predicado `invierte(L, R)` que se satisface cuando `R` contiene los elementos de `L` en orden inverso. Utiliza el predicado `concatena/3` (/n: indica n argumentos):

```
concatena([], L, L).
concatena([X|L1], L2, [X|L3]) :-
    concatena(L1, L2, L3).
```

que se satisface cuando su tercer argumento es el resultado de concatenar las dos listas que se dan como primer y segundo argumento.

Ejemplos:

```
?- concatena([], [1, 2, 3], L).
L = [1, 2, 3].

?- concatena([1, 2, 3], [4, 5], L).
L = [1, 2, 3, 4, 5].

?- invierte([1, 2], L).
L = [2, 1].

?- invierte([], L).
L = [].

?- invierte(L, [1, 2]).
L = [2, 1].
```

EJERCICIO 3 (1 punto). Implementar el predicado palindromo(L) que se satisface cuando L es una lista palíndroma, es decir, que se lee de la misma manera de izquierda a derecha y de derecha a izquierda.

```
?- palindromo([1, 2, 1]).
true.

?- palindromo([1, 2, 1, 1]).
false.
```

¿Qué pasa si se llama palindromo(L), donde L es una variable no instanciada? Explicar.

EJERCICIO 4 (1 punto). Implementar el predicado divide(L,N,L1,L2) que se satisface cuando la lista L1 contiene los primeros N elementos de L y L2 contiene el resto.

```
?- divide([1, 2, 3, 4, 5], 3, L1, L2).
L1 = [1, 2, 3],
L2 = [4, 5]

?- divide(L, 3, [1, 2, 3], [4, 5, 6]).
L = [1, 2, 3, 4, 5, 6].
```

EJERCICIO 5 (1 punto). Implementar el predicado `aplasta(L, L1)` que se satisface cuando la lista `L1` es una versión "aplastada" de la lista `L`, es decir, si uno de los elementos de `L` es una lista, esta será remplazada por sus elemento, y así sucesivamente.

```
?- aplasta([1, [2, [3, 4], 5], [6, 7]], L)
L = [1, 2, 3, 4, 5, 6, 7].

?- aplasta(L,[1, 2, 3])
¿Qué pasa si se hace esto? Explicar.
```

EJERCICIO 6 (1 punto). Implementar el predicado `primos(N, L)` que se satisface cuando la lista `L` contiene los factores primos del número `N`

```
?- primos(100,L).
L = [2, 2, 5, 5].
```

No es necesario que la función funcione en los dos sentidos. Una implementación perfectamente aceptable proporciona:

```
?- primos(N,[2, 2, 5, 5]).
>/2: Arguments are not sufficiently instantiated
```

Sugerencia: Para esta función puede ser útil crear el predicado next_factor(N,F,NF), que genera los factores que vamos a probar. Dado el número n y el último factor generado, el predicado se satisface si:

1. F=2 y NF=3, o
2. F<sqrt(N) y NF=F+2

(no hace falta probar los número pares, excepto 2).

EJERCICIO 7 (2 puntos). Codificación run-length de una lista: si la lista contiene N términos consecutivos iguales a X, esto son codificados como un par [N, X]. Ejemplo:

[1, 1, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 7, 7] se codifica como [[2, 1], [1, 2], [2, 3], [3, 4], [4, 5], [1, 6], [2, 7]]

EJERCICIO 7.1 Empezamos con el predicado cod_primero(X, L, Lrem, Lfront). Lfront contiene todas las copias de X que se encuentran al comienzo de L, incluso X; Lrem es la lista de elementos que quedan:

```
?- cod_primero(1, [1, 1, 2, 3], Lrem, Lfront)
Lrem = [2, 3],
Lfront = [1, 1, 1];
false.

?- cod_primero(1, [2, 3, 4], Lrem, Lfront)
Lrem = [2, 3, 4],
Lfront = [1];
false.
```

EJERCICIO 7.2 El predicado cod_all(L, L1) aplica el predicado cod_primero/4 a toda la lista L.

```
?- pack([1, 1, 2, 3, 3, 3, 3], L).
L = [[1, 1], [2], [3, 3, 3, 3]];
false.
```

EJERCICIO 7.3 El predicado run_length(L, L1) aplica el predicado pack y luego transforma cada una de las listas en la codificación run-length:

```
?- run_length([1, 1, 1, 1, 2, 3, 3, 4, 4, 4, 4, 4, 5, 5], L).
L = [[4, 1], [1, 2], [2, 3], [5, 4], [2, 5]];
false.
```

Un árbol de Huffman es una estructura usada para **codificar y comprimir información**. El objetivo es codificar un texto formado por símbolos utilizando códigos binarios. El esquema de codificación utiliza la frecuencia que aparece cada uno de los símbolos en un texto para minimizar el número de bits necesario para almacenar la información. En concreto,los símbolos más frecuentes en el texto están codificados con bloques de bits más cortos,. A modo de ejemplo, la cadena en ASCII,

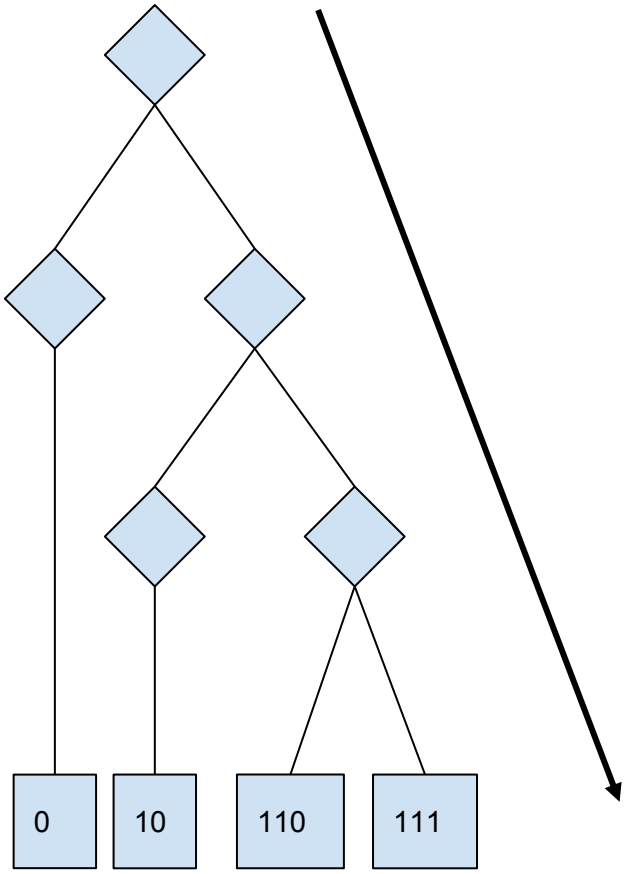
```
AAAADAAACCCAAAAAAABAAAAABAAA
```

usa para almacenar cada caracter 1 Byte. Sin embargo, podemos reducir el espacio ocupado por la cadena. De forma simple podemos expresarlo como:

A	0
B	111
C	10
D	110

De este modo, utilizando los códigos ASCII por los códigos de bloques de bits especificados en esta tabla, es posible reducir la longitud de la cadena sin perder información. Es importante observar que ningún código es prefijo de otro código (por ejemplo, solo el código del carácter ‘A’, el más frecuente en la cadena, empieza por ‘0’. De forma análoga, solo el código del carácter ‘C’ comienza por ‘10’, etc.), por lo que no hay ambigüedad al descodificar.
En esta práctica implementaremos una versión simplificada de este tipo de estructura.

Para ello insertaremos el elemento con mayor probabilidad a la izquierda del nodo raíz, y seguiremos con este mismo proceso expandiendo el árbol por la derecha. Cuando solo queden 2 elementos dejaremos de expandir el árbol y terminaremos como se muestra en la figura.



EJERCICIO 8 (2 puntos). Implementa el predicado `build_tree(List, Tree)` que transforma una lista de pares de elementos ordenados en una versión simplificada de un árbol de Huffman. Para representar árboles usaremos las funciones `tree(Info, Left, Right)` y `nil`. También usaremos el predicado `concatena/3`. Ejemplo:

```
?-build_tree([p-0, a-6, g-7, p-9, t-2, 9-99], X).
X = tree(1, tree(p, nil, nil), tree(1, tree(a, nil, nil), tree(1, tree(g, nil, nil),
tree(1, tree(p, nil, nil), tree(1, tree(t, nil, nil), tree(9, nil, nil))))))
false

?-build_tree([p-55, a-6, g-7, p-9, t-2, 9-99], X).
X = tree(1, tree(p, nil, nil), tree(1, tree(a, nil, nil), tree(1, tree(g, nil, nil),
tree(1, tree(p, nil, nil), tree(1, tree(t, nil, nil), tree(9, nil, nil))))))
False

?-build_tree([p-55, a-6, g-2, p-1], X).
X = tree(1, tree(p, nil, nil), tree(1, tree(a, nil, nil), tree(1, tree(g, nil, nil),
tree(p, nil, nil))))
False

?-build_tree([a-11, b-6, c-2, d-1], X).
X = tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1, tree(c, nil, nil),
tree(d, nil, nil))))
```

Los nodos hoja del árbol se corresponden con los elementos de la lista ordenada y almacenan el elemento en el campo `Info`, mientras que los nodos intermedios siempre almacenan un 1.

EJERCICIO 8.1. Implementar el predicado `encode_elem(X1, X2, Tree)` que codifica el elemento (`X1`) en (`X2`) basándose en la estructura del árbol (`Tree`). Así tomando el árbol del ejemplo anterior

```
?-build_tree([a-11, b-6, c-2, d-1], X).
X = tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1, tree(c, nil, nil),
tree(d, nil, nil))))
```

Tenemos los siguientes ejemplos:

```
?- encode_elem(a, X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1,
tree(c, nil, nil), tree(d, nil, nil))))).
X = [0]
false

?- encode_elem(b, X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1,
tree(c, nil, nil), tree(d, nil, nil))))).
X = [1, 0]
false

?- encode_elem(c, X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1,
tree(c, nil, nil), tree(d, nil, nil))))).
X = [1, 1, 0]
false

?- encode_elem(d, X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1,
tree(c, nil, nil), tree(d, nil, nil))))).
X = [1, 1, 1]
false
```

EJERCICIO 8.2. Implementar el predicado `encode_list(L1, L2, Tree)` que codifica la lista (`L1`) en (`L2`) siguiendo la estructura del árbol (`Tree`). Ejemplos:

```
?- encode_list([a], X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1,
tree(c, nil, nil), tree(d, nil, nil))))).
X = [[0]]
false

?- encode_list([a,a], X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil), tree(1,
tree(c, nil, nil), tree(d, nil, nil))))).
X = [[0], [0]]
false

?- encode_list([a,d,a], X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil),
tree(1, tree(c, nil, nil), tree(d, nil, nil))))).
X = [[0], [1, 1, 1], [0]]
false
```

```
?- encode_list([a,d,a,q], X, tree(1, tree(a, nil, nil), tree(1, tree(b, nil, nil),
tree(1, tree(c, nil, nil), tree(d, nil, nil))))).
false.
```

EJERCICIO 8.3. Implementar el predicado `encode(L1, L2)` que codifica la lista (`L1`) en (`L2`). Para ello haced uso del predicado `dictionary`

```
dictionary([a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]).
```

Ejemplos:

```
?- encode([i,n,t,e,l,i,g,e,n,c,i,a,a,r,t,i,f,i,c,i,a,l],X).
X = [[0], [1, 1, 1, 0], [1, 1, 0], [1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 0], [0], [1, 1, 1,
1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 0], [1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 0], [0], [1,
0], [1, 0], [1, 1, 1, 1, 1, 1, 1, 0], [1, 1, 0], [0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
[0], [1, 1, 1, 1, 1, 1, 0], [0], [1, 0], [1, 1, 1, 1, 0]]
False

?- encode([i,a],X).
X = [[0], [1, 0]]
False

?- encode([i,2,a],X).
false
```