Fecha 2 de Junio de 2016

Práctica 2: Programación Lógica Pura

MEMORIA DE LA PRÁCTICA
ETSIINF UPM

- Fernández Castro, Roberto Daniel - 110069
- Alobuela Collaguazo, Maria Jose - 130227

Índice de Títulos

Introducción

En esta memoria se explicará la solución que proponemos a la segunda práctica de la asignatura, en la que se emplea programación ISO prolog para su realización. Dicha práctica está dividida en dos partes.
Por un lado, tenemos la primera parte en la que se pide programar el predicado enumerar/3, el cual enumera los nodos y los arcos de un grafo conexo bajo ciertas condiciones.
Por otro lado, en la segunda parte se pide programar el predicado reparación/3, el cual devuelve el orden en el que deben trabajar un conjunto de buzos para evitar el hundimiento total del buque, para solucionar el problema empleamos backtraking .

En el documento se empezará mostrando y explicando los predicados que han sido utilizados para la realización de las correspondientes partes de la práctica. A continuación se mostrará tanto para la primera como para la segunda parte de la práctica el código con las correspondientes explicaciones, luego se mostrarán las consultas que hemos realizado para comprobar el correcto funcionamiento del código y por último unas conclusiones finales.

Parte Uno: Problema de enumeración

Programa Principal

Predicado enumerar/3.

El siguiente predicado enumera los nodos y los arcos de un grafo conexo, que está representado por una lista de pares de la forma a-b. Si hay N nodos, se realiza la enumeración asociando cada nodo con un número de 1 a N (distinto para cada nodo) y cada arco con un número de 1 a N-1 (distinto para cada arco), de tal forma que, además, cada arco A-B asociado al número K es tal que la diferencia entre los números asociados a A y B (en valor absoluto) es exactamente K.
Los argumentos del predicado son:

- Arco: Argumento de tipo lista (parámetro de entrada).
- EnumNodos: Argumento de tipo lista (parámetro de salida).
- EnumArcos: Argumento de tipo lista (parámetro de salida).

```
enumerar(Arco, EnumNodos, EnumArcos):-
      length(Arco,Long),
      findall(X,between(1,Long+1,X),Xs),
      position_node(Xs,[],L),
      database(Arco,L),!,
      findall(elem(K,V),nodo(K,V),EnumNodos),
      findall(elem(K,V),arco(K,V),EnumArcos),
      retractall(nodo(_,_)),
      retractall(arco(_,_)).
```

Para realizar la implementación anterior utilizamos los siguientes predicados:

Predicado length/2.

Calcula la longitud de la lista del grafo(Arco, parámetro de entrada) devolviéndolo en Long (parámetro de salida).

Predicado findall/3.

Nos devuelve la lista Xs (parámetro de salida) con los números entre 1 y Long+1 (generados por el predicado between/3) obteniendo así, el número de nodos que contiene el grafo.

Predicado position_node/3:

El siguiente predicado trata la lista Xs (parámetro de entrada) para devolvernos otra lista (parámetro de salida) con la secuencia ordenada de números que le corresponde a cada nodo
Los argumentos del predicado son:

- Xs: Argumento de tipo lista.
- [ ]:Argumento de tipo lista.
- L: Argumento de tipo lista.

Ejemplo de lo que hace position_node:

Xs=[1,2,3], L=[3,1,2]
Xs=[1,2,3,4,5], L=[5,1,4,2,1]

```
position_node([],Acc,Acc).
position_node([First|Y],Acc,L):-
      last([First|Y],Last),
      First\=Last,
      append(Acc,[Last,First],Acc2),
      delete(Y,Last,L1),
      position_node(L1,Acc2,L).
position_node([First|Y],Acc,L):-
      append(Acc,[First],L),
      position_node(Y,L,L).
```

Para ello, utilizamos tres predicados dentro de position_node:

Predicado last/2.

Selecciona el último elemento de la lista para insertarlo en la nueva lista(L).

Predicado append/3.

Concatena los elementos de la lista en una nueva lista de manera que el último elemento de Xs será el primer elemento de la nueva lista y el primer elemento de Xs se insertará después, así sucesivamente. Si el elemento de la lista Xs llega a ser el primero y último(se verifica en el segundo append) se inserta en la nueva lista, detrás de todos los elementos ya insertados.

Predicado delete/3.

Borra el primer elemento de la lista para que en la siguiente iteración el segundo elemento sea el primero de la lista.

Predicado database/2.

El siguiente predicado recorre la lista que contiene las aristas del grafo(Arco, parámetro de entrada), para insertar en la base de hechos por un lado, los nodos con el número que le corresponda de la lista L (parámetro de salida) y por otro los arcos, mediante el predicado hashmap.
Los argumentos del predicado son:

- Arco: Argumento de tipo lista.
- L: Argumento de tipo lista

```
database([],_).
database([X-Y|Z],L):-
      hashmap(X,L,L1),
      hashmap(Y,L1,L2),
      absolute(X,Y,V),
      assert(arco(X-Y,V)),
database(Z,L2).
```

Los predicados empleados en database/2 son los siguientes:

Predicado hashmap/2.

El siguiente predicado inserta los nodos de manera que no se repitan, es decir, si el nodo ya se encuentra en la base hechos no se vuelve a insertar, de esta manera cada nodo se une con su correspondiente número. Para insertar el nodo (K,V) en la base de hechos empleamos el predicado assert.
Los argumentos del predicado son:

- Arco: Argumento de tipo lista (parámetro de entrada).
- L: Argumento de tipo lista (parámetro de entrada).

```
hashmap(K,[V|T],L):-
      \+(nodo(K,_)),
      assert(nodo(K,V)),
      L=T.
hashmap(_,L,L).
```

Predicado absolute/3.

Calcula la diferencia en valor absoluto de los números asociados a dos nodos.

```
absolute():-
      nodo(X,V1),
      nodo(Y,V2),
      Z is abs(V1-V2).
```

Predicado findall/3.

Con el siguiente predicado obtenemos de la base de hechos la lista de nodos en EnumNodos y la lista de arcos en EnumArcos.

Predicado retractall/1.

Elimina de la base de hechos tanto los nodos como los arcos, de manera que al insertar otra lista de arcos se podrá llevar a cabo el predicado enumerar correctamente, pues los valores calculados anteriormente habrán sido borrados.

Consultas Realizadas para el problema de enumeración.

En las siguientes pruebas, el argumento de entrada se corresponde con una lista que contiene los arcos de un nodo, el resultado de llamar al predicado enumerar, será obtener dos listas (argumentos de salida): EnumNodos, que contendrá los nodos del grafo con el número correspondiente y EnumArcos, que contendrá los arcos del grafo.

```
?- enumerar([a-b,b-c],EnumNodos,EnumArcos).

EnumArcos = [enum(2,a-b),enum(1,b-c)],
EnumNodos = [enum(3,a),enum(1,b),enum(2,c)] ?.

no.

?- enumerar([a-b,b-c,c-d,d-e],EnumNodos,EnumArcos).

EnumArcos = [enum(4,a-b),enum(3,b-c),enum(2,c-d),enum(1,d-e)],
EnumNodos = [enum(5,a),enum(1,b),enum(4,c),enum(2,d),enum(3,e)] ?.

no.
```

Parte Dos: Accidente Marítimo

Programa Principal

Predicado reparación/3

El siguiente predicado evita el hundimiento total del buque, para ello, los buzos(Buzo(B,T)) asignados para su reparación trabajan conjuntamente en un determinado tiempo de manera que el trabajo de todos ellos, no supere el tiempo en el que el buque se hundiría totalmente. Primero obtenemos todas las posibles soluciones que podrían hacerse con la lista de buzos pasada como parámetro de entrada, a continuación verificamos que cada buzo no haya trabajado con otro antes.
Los argumentos del predicado son:

- Equipo: Argumento de tipo lista (argumento de entrada).
- Tiempo: Argumento de tipo entero(argumento de entrada).
- OrdenParejas: Argumento de tipo lista(argumento de salida).

```
reparacion(Equipo,Tiempo,OrdenParejas):-
      permutation(Equipo,Backtraking),
      length(Equipo,Len),
      Leng is Len - 1,
      numlist(0,Leng,NumPareja),
      sumlist(NumPareja,LenPareja),
      immerse(Backtraking,Backtraking,LenPareja),
      findall(T,pareja(_,_,T), LT),
      sumlist(LT,Total),
      timecheck(Total,Tiempo),

findall(pareja(X,Y,Z),pareja(X,Y,Z),OrdenParejas),
      retractall(pareja(_,_,_)).
```

Para la realización del predicado, empleamos otros predicados:

Predicado permutation/2.

El siguiente predicado permuta la lista con todas las posibles combinaciones de parejas de buzos.
Los argumentos del predicado son:

- Equipo: Argumento de tipo lista (argumento de entrada).
- Backtraking: Argumento de tipo lista (argumento de entrada).

```
permutation(Bs,[A|As]):-
      append(Xs,[A|Ys],Bs),
      append(Xs,Ys,Zs),
      permutation(Zs,As).
permutation([],[]).
```

Predicado inmerse/3.

Este predicado va iterando la lista de buzos, insertando en caso de que aún no hayan trabajado juntos. Va modificando los tiempos de inmersión de los buzos para poder insertar el tiempo que han trabajado los dos. Acaba cuando han insertado todas las posibles parejas de los buzos.
Los argumentos del predicado son:

- Bactraking: Argumento de tipo lista (argumento de entrada).
- Backtraking: Argumento de tipo lista (argumento de entrada).
- LenPareja: La variable LenPareja es el número de parejas posibles según el tamaño del equipo.

```
immerse([buzo(B1,T1),buzo(B2,T2)|R],L,Length) :-
      T1=T2,
      insert(B1,B2,T1,[buzo(B1,T1),buzo(B2,T2)|R],L,Length),
      immerse(R,L,Length),!.
immerse([buzo(B1,T1),buzo(B2,T2)|R],L,Length) :-
      (T1 < T2 -> T = T1, B = B2 ; T = T2, B = B1),
      insert(B1,B2,T,[buzo(B1,T1),buzo(B2,T2)|R],L,Length),
      TR is abs(T1 -T2),
      immerse([buzo(B,TR)|R],L,Length),!.
immerse(_,_,Length):-
      findall(pareja(X,Y,Z),pareja(X,Y,Z),L),
      length(L,R),
      R =:= Length,!.
immerse([buzo(B1,T1)],List,Length):-
      deldiver(buzo(B1,_),List,Del),
      append([buzo(B1,T1)],Del,NewList),
      immerse(NewList,Del,Length),!.
immerse([],List,Length):-
      immerse(List,List,Length),!.
```

Predicado deldiver/3

El siguiente predicado elimina un buzo de la lista de buzos.
Los argumentos del predicado son:

- buzo(B,_) : Argumento de tipo buzo, corresponde al buzo que queremos eliminar de la lista (Argumento de entrada).
- [ ]: Argumento de tipo lista, donde se encuentran los buzos (Argumento de entrada).
- [ ]:Argumento de tipo lista(Argumento de salida).

```
deldiver(buzo(B,_),[buzo(B,_)|Tail],Tail):-!.
      deldiver(buzo(B1,_),[buzo(B2,T2)|Tail],[buzo(B2,T2)|Tail1]):-
      deldiver(buzo(B1,_),Tail,Tail1).
```

Predicado insert/6

Inserta las parejas de buzos en caso de que no hayan trabajado juntos. Si lo han hecho, itera la lista de buzos, para probar otra combinación.
Los argumentos del predicado son:

- B1: Nombre del Buzo 1 (argumento de entrada).
- B2: Nombre del Buzo 2 (argumento de entrada).
- T: Tiempo que han trabajado juntos (argumento de entrada).
- [ ]: Argumento lista de buzos. (argumento de entrada).
- L: Argumento lista de buzos. (argumento de entrada).
- Length: Número de posibles parejas según la lista (argumento de entrada).

```
insert(B1,B2,T,_,_,_):-
      \+(pareja(B1,B2,_)),
      \+(pareja(B2,B1,_)),
      assert(pareja(B1,B2,T)).
insert(_,_,_,[buzo(B1,T1),buzo(_,_)|R],L,Length):-
      immerse([buzo(B1,T1)|R],L,Length).
```

Predicado sumList/2.

El predicado se encarga de sumar los elementos de una lista.
Los argumentos del predicado son:

- [ ]: Argumento de tipo lista (argumento de entrada).
- S: Argumento de tipo entero (argumento de salida).

```
sumList([],0).
sumList([X|Xs],S):-
      sumList(Xs,S2),
      S is S2 + X.
```

Predicado timecheck/2.

Comprueba si el tiempo en el que trabajan todos los buzos conjuntamente no supera el tiempo máximo pasado como argumento, si no es así, falla y realiza backtraking para dar otra posible solución.
Los argumentos del predicado son:

- Time: Argumento de tipo entero (argumento de entrada).
- Max: Argumento de tipo entero (argumento de entrada).

```
timecheck(Time,Max):- Time =< Max,
timecheck(_,_):- retractall(pareja(_,_,_)), fail.
```

Consultas Realizadas para accidente marítimo/3

A continuación, realizamos algunas pruebas para comprobar que el predicado funciona correctamente, nos da distintas soluciones que corresponden a las distintas parejas que pueden formar los buzos sin superar el tiempo máximo.

```
?-
reparacion([buzo(gomez,45),buzo(lopez,20),buzo(garcia,40),buzo(perez,15)
], 80, OrdenParejas).

OrdenParejas =
[pareja(gomez,lopez,20),pareja(gomez,perez,15),pareja(gomez,garcia,10),p
areja(garcia,lopez,20),pareja(garcia,perez,10),pareja(perez,lopez,5)] ?
;

OrdenParejas =
[pareja(gomez,perez,15),pareja(gomez,lopez,20),pareja(gomez,garcia,10),p
areja(garcia,perez,15),pareja(garcia,lopez,15),pareja(lopez,perez,5)] ?
;

...
```

Conclusiones

En conclusión, estamos muy contentos con este proyecto, ha sido un reto intelectual, ya que básicamente ambos ejercicios eran algorítmicos y resolverlos ha sido satisfactorio. Por otro lado hemos mejorado los conocimientos de la programación declarativa.