Université d'Oran 1 Ahmed Ben Bella
Module Programmation Logique TP
Année universitaire 2017/2018

Oran, Décembre 2017

Projet Prolog TP :

Application de la programmation logique sur les graphes avec SWI-Prolog

Travail réalisé par :

- CHOUIB Chawki (groupe 1)
- DJEZIRI Ibtissem (groupe 1)

Table des matières :

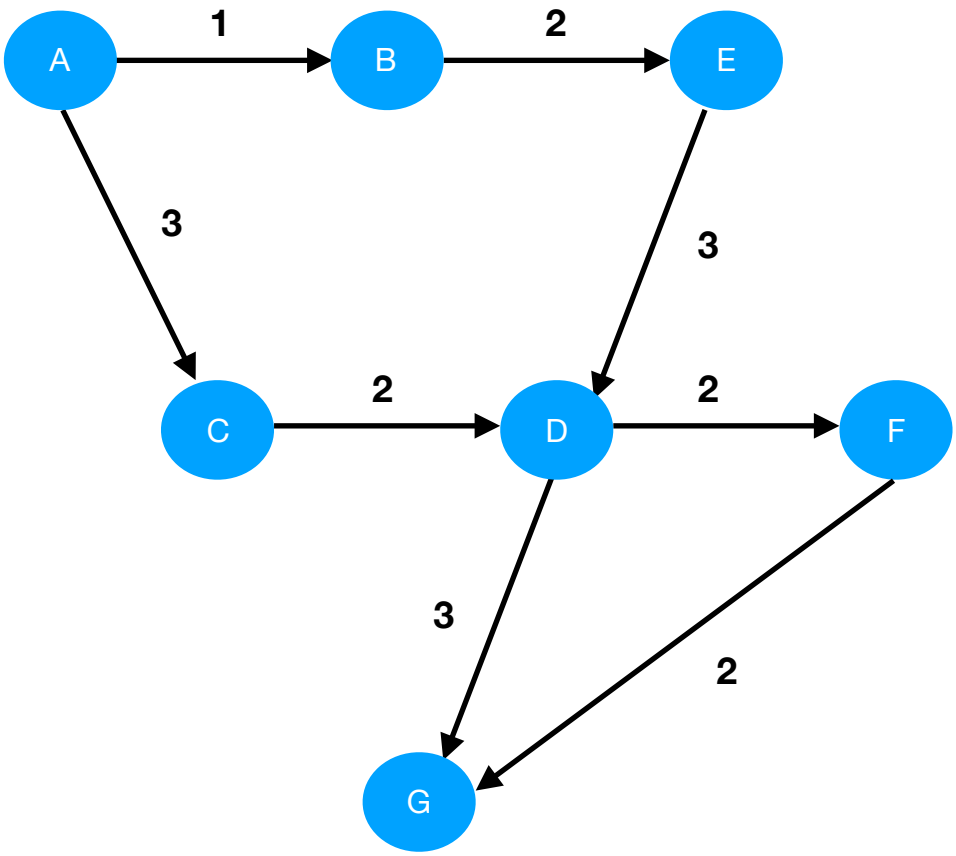
I. Enoncer

- Reprendre en utilisant les listes le TP sur les graphes orienté :
  - Teste si il existe un circuit (cycle dans le cas d'un graphe non orienté).
  - Teste si le graphe est connexe.
  - Afficher le chemin le plus court entre 2 sommet quelconque (moindre cout, algorithme de Dijkstra).
  - Afficher tous les chemins possible entre 2 sommet quelconque.

II. Réponse aux questions

- Pour répondre aux questions voici un graphe G orienté tel que G = (V , A) ' V ' l'ensemble de sommet et ' A ' l'ensemble d'arcs

  - V = { A, B, C, D, E, F, G }.
  - A = { (A,B), (B,E), (A,C), (C,D), (E,D), (D,F), (D,G) }



1. Représentation des sommets du graphe

```
% les sommets du graph
sommet(a).
sommet(b).
sommet(c).
sommet(d).
sommet(e).
sommet(f).
sommet(g).
```

Remarque : Dans SWI-Prolog tous les noms en majuscule c'est des variables, or nos sommets c'est des constante, donc la représentation des sommets serra faite par les lettres minuscule.

2. Teste sommet existant dans le graphe

```
% predicat test sommet existe ou non.
isSommet(Point):- sommet(Point).
```

- Retourne **True** si sommet existe dans les fais, sinon **False**.

3. Représentation des arcs du graphe

```
% les arcs (couple de sommet avec cout).
arc(a,b,1).
arc(b,e,2).
arc(a,c,3).
arc(c,d,2).
arc(e,d,3).
arc(d,f,2).
arc(d,g,3).
arc(f,g,2).
```

4. Teste chemin existant entre 2 sommets quelconque

- SANS LISTE :

```
% predicat pour test un chemin si il existe .
chemin(Point1,Point2):- arc(Point1,Point2,_),!.

chemin(Point1,Point2):- arc(Point1,PointT,_),chemin(PointT,Point2).
```

- Si on trouve un arc correspondant au 2 points, on arrête et on retourne **True** sinon on fait un appel récursive jusqu'à l'obtention du chemin voulu sinon on retourne **False.**

- AVEC LISTE :

```
% predicat pour tester un chemin si il existe avec une list.
chemin2(Point1,Point2):- chemin2(Point1,Point2,[]).

chemin2(Point1,Point2,_):- arc(Point1,Point2,_),!.

chemin2(Point1,Point2,List):-  not(member(Point1,List)),
                               arc(Point1,PointT,_),
                               chemin2(PointT,Point2,[Point1|List]).
```

5. Teste existence de circuit dans le graphe

```
% dans un graph oriente en parle de circuit , pas de cycle
% on appelle circuit une suite d'arcs consecutifs (chemin) dont les deux
% sommets extremites sont identiques.
testCircuit(Point):- chemin2(Point,Point),
                     write('Graphe avec circuit '),!.

testCircuit(Point):- write('Graphe sans circuit '),fail.
```

- Dans un graphe oriente en parle de circuit , pas de cycle, on appelle circuit une suite d'arcs consécutifs (chemin) dont les deux sommets extrémités sont identiques.
- Donc avec le prédicat **chemin/2** on teste et si on trouve un chemin d'un point à lui même , si existe retourne True sinon retourne False.

6. Teste si le graphe est connexe

```
% Le graphe est dit connexe lorsqu'il existe une chaine entre deux
% sommets quelconques.
testConnexe(Point1,Point2):- chemin2(Point1,Point2),
                             chemin2(Point2,Point1),
                             write('Graph connexe '),!.

testConnexe(Point1,Point2):- write('Graph non connexe '),fail.
```

- Le graphe est dit connexe lorsqu'il existe une chaîne entre deux sommets quelconques c'est-a-dire un chemin de Point1 a Point2 et Point2 a Point1.
- Retourne un message dans le cas **True** et **False**.

7. Afficher tous les chemins possible entre 2 sommets

```
% afficher tous les chemins possible entre deux sommets dans des liste
% avec le cout total de chaque chemin
afficheAllChemin(Point1,Point2):- not(chemin2(Point1,Point2)),
                                  write('Aucun chemin'),!,fail.

afficheAllChemin(Point1,Point2):- forall(afficheAllChemin(Point1,Point2,Cout,Chemin,[]),
                                  ( write('----> Cout : '), write(Cout),
                                  write('    Chemin : '), write(Chemin),nl )).

afficheAllChemin(Point1,Point2,Cout,[Point1,Point2],_) :- arc(Point1,Point2,Cout).

afficheAllChemin(Point1,Point2,Cout,[Point1|List1],List2):- not(member(Point1,List2)),
                                                            arc(Point1,PointT,CoutT),
                                                            afficheAllChemin(PointT,Point2,Cout2,List1,[Point1|List2]),
                                                            Cout is Cout2+CoutT.
```

- **forall(+prédicat, +action)** est utiliser pour éviter d'appuyer a chaque fois sur le point virgule lors de l'exécution du prédicat **afficheAllChemin/4.**
- Dans notre cas, **forall** aura:
    - Prédicat c'est **afficheAllChemin/4**
    - Action c'est **write()**.
- **AfficheAllChemin/4** a le même principe que **chemin2/3** sauf que la on a rajouter la somme des couts a chaque appel récursive du prédicat.

8. Afficher le chemin le plus cour (qui a le cout le plus petit)

```
% afficher le chemin le plus court  (le plus court chemin).
:-dynamic(cheminMin/2).

afficheMinChemin(Point1,Point2):- not(chemin2(Point1,Point2)),write('Aucun chemin'),!,fail.

afficheMinChemin(Point1,Point2):- afficheMinChemin(Point1,Point2,Cout,Chemin),
                                  (write('Cout : '), write(Cout),
                                  write('  Chemin : '),write(Chemin),nl).

afficheMinChemin(Point1,Point2,Cout,Chemin):- not(cheminMin(_,_)),
                                              afficheAllChemin(Point1,Point2,CoutT,CheminT,[]),
                                              assertz(cheminMin(CoutT,CheminT)),!,
                                              afficheMinChemin(Point1,Point2,Cout,Chemin).

afficheMinChemin(Point1,Point2,_,_):- afficheAllChemin(Point1,Point2,CoutT2,CheminT2,[]),
                                      cheminMin(CoutT,CheminT),CoutT2 < CoutT,
                                      retract(cheminMin(CoutT,CheminT)),
                                      asserta(cheminMin(CoutT2,CheminT2)),fail.

afficheMinChemin(_,_,Cout,Chemin):- cheminMin(Cout,Chemin),
                                    retract(cheminMin(Cout,Chemin)).
```

- Par défaut, un prédicat est statique.
- Un prédicat qu'on veut pouvoir manipuler dynamiquement doit être déclaré comme tel avec la directive **dynamic/1**.
- Dans notre cas c'est **:-dynamic(cheminMin/2)**.
- On va afficher tous les chemin possible entre 2 sommets grâce au prédicat **afficheAllChemin/4**
- On considère que le premier chemin c'est le min et on l'insert dans le prédicat dynamique **cheminMin/2** avec le prédicat **assertz/1.**
  - ***assertz/1*** *insère la clause en queue de programme*
- Et on fait un appel récursive du prédicat **afficheMinChemin/4.**
- Si le chemin suivant a un coup plus petit que celui enregistré dans le prédicat dynamique alors on le retire avec **retract/1** dans la 3eme règle.
  - ***retract/1*** *retire le premier fait rencontré qui s'unifie avec le prédicat.*
- Et on insert le nouveau chemin le plus cour avec **asserta/1**.
  - ***asserta/1*** *insère la clause en tête du programme.*

- Donc on remplir la base de données Prolog avec la première solution, puis de procéder à la collecte et à la comparaison des solutions multiples sous-jacentes afin de toujours laisser la meilleure solution dans la base de données Prolog.

9. Exemple d'exécution

TESTE SOMMET EXISTANT :

```
% /Users/chawki/Prolog Projects/hhhprojet compiled 0.01 sec, 6 clauses
?- isSommet(X).
X = a
X = b
X = c
X = d
X = e
X = f
X = g.

?- isSommet(g).
true.

?- isSommet(x).
false.
```

TESTE CHEMIN :

```
?- chemin2(a,g).
true .

?- chemin2(g,a).
false.
```

TESTE CIRCUIT :

```
?- testCircuit(X).
Graphe sans circuit
false.
```

TESTE CONNEXE :

```
?- testConnexe(X,Y).
Graph non connexe
false.
```

AFFICHER TOUS LES CHEMINS D'UN POINT A UN AUTRE POINT :

```
?- afficheAllChemin(a,g).
----> Cout : 9    Chemin : [a,b,e,d,g]
----> Cout : 10    Chemin : [a,b,e,d,f,g]
----> Cout : 8    Chemin : [a,c,d,g]
----> Cout : 9    Chemin : [a,c,d,f,g]
true.
```

AFFICHER LE CHEMIN LE PLUS COUR D'UN POINT A UN AUTRE POINT :

```
?- afficheAllChemin(a,g).
----> Cout : 9    Chemin : [a,b,e,d,g]
----> Cout : 10    Chemin : [a,b,e,d,f,g]
----> Cout : 8    Chemin : [a,c,d,g]
----> Cout : 9    Chemin : [a,c,d,f,g]
true.

?- afficheMinChemin(a,g).
Cout : 8  Chemin : [a,c,d,g]
true.
```