Université d'Orléans **M1 Informatique**
Programmation en logique et par contraintes **Année 2011-2012**

Contrôle continu du 4 novembre 2011

Durée : 2h. Documents autorisés : notes de cours et de TD. Expliquez et commentez bien vos programmes. Le sujet est sur 2 pages. Barème à titre indicatif : $5 + 4 + 11$

Exercice 1 Soit `concat/3` le prédicat de concaténation de listes. Il peut être défini de la manière suivante :

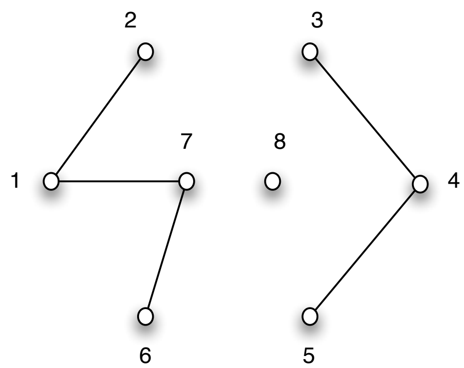
```
concat([], L, L).
concat([X|L], M, [X|N]) :- concat(L, M, N).
```

1. Dressez l'arbre de recherche produit par le but `concat(L, M, [1,2,3])` en exhibant les substitutions produites en cas de succès.
2. Sur ce même arbre, représentez (simplement avec une autre couleur) l'effet du *cut* si la première clause était donnée ainsi : `concat([], L, L) :- !.`

Exercice 2 Soit `consecutifs(X, Y, L)` le prédicat qui est vrai si les éléments `X` et `Y` sont consécutifs (dans cet ordre) dans la liste `L`. Donnez **deux** versions qui implémentent ce prédicat : une avec utilisation du prédicat `concat/3`, l'autre sans.

Exercice 3 Composants connexes dans un graphe.

On considère un graphe non orienté de N sommets, numérotés de 1 à N. À chaque sommet est associée une liste de numéros des sommets adjacents. Le graphe est alors représenté par une liste de N listes de sommets adjacents. Par exemple le graphe suivant est représenté par la liste `[[2,7],[1],[4],[3,5],[4],[7],[1,6],[]]`



Un composant connexe du graphe est une liste de sommets telle qu'il existe un chemin reliant deux sommets quelconques du composant. Le graphe précédent a trois composants connexes, qui sont `[1,2,6,7]`, `[3,4,5]` et `[8]`. L'objectif est d'écrire un prédicat `cc(+G)`, qui affiche la liste des composants connexes du graphe G. L'algorithme utilisé sera comme suit :

- Soit N le nombre de sommets du graphe. Au début, on considère que chaque sommet est dans un composant connexe séparé. Pour $N = 8$, la liste de composants connexes initiale est donc `[[1], [2],...,[8]]`.
- Pour chaque sommet u, on prend le composant connexe C_u qui contient u. Pour chaque sommet v dans la liste des sommets adjacents à u, si v n'est pas dans C_u alors on fusionne C_u et le composant connexe C_v qui contient v en un seul.

Afin de réaliser cet algorithme, répondez aux questions suivantes. Les questions sont indépendantes, vous pouvez répondre à une questions en utilisant des prédicats des autres questions même s'ils ne sont pas faits. Ecrivez des prédicats auxiliaires si besoin. Vous pouvez utiliser des prédicats pré-définis de Prolog, en ce cas respectez la spécification de ces prédicats.

1. Ecrire un prédicat `nbSommets(+G, -N)` qui calcule le nombre N de sommets de G.
2. Ecrire un prédicat `creerListe(+N,-L)`, qui crée une liste `L = [[1],...,[N]]`.
3. Ecrire un prédicat `trouver(+U, +LC1, -Cu, -LC2)` qui trouve le composant connexe C_u qui contient le sommet U dans la liste de composants connexes $LC1$, et crée la liste $LC2$ à partir de $LC1$ en enlevant C_u.

   ```
   ?- trouver(2, [[3,4],[2,1,6],[5],[7],[8]], C, LC).
   C = [2,1,6]
   LC = [[3,4],[5],[7],[8]]
   ```

4. Ecrire un prédicat `fusion(+L1, +L2, -L3)` qui calcule une liste $L3$ qui est la fusion de deux listes $L1$ et $L2$.
5. Ecrire un prédicat `parcour1liste(+C, +Ladj, +LC1, -LC2)`, qui, pour chaque élément v de la liste $Ladj$ qui n'est pas dans C, trouve le composant connexe C_v, le sort de $LC1$ et le fusionne avec C. La liste $LC2$ est une nouvelle liste de composants connexes après ces fusions.

   ```
   ?- parcour1liste([1], [2,7], [[2],[3],[4],[5],[6],[7],[8]], L).
   L = [[1,2,7],[3],[4],[5],[6],[8]]

   ?- parcour1liste([3,4], [3,5], [[1,2,7],[5],[6],[8]], L).
   L = [[3,4,5],[1,2,7],[6],[8]]

   ?- parcour1liste([8], [], [[1,2,7,6],[3,4,5]], L).
   L = [[8],[1,2,7,6],[3,4,5]]
   ```

6. Ecrire un prédicat `compconnexes(+G, -L)`, qui calcule L la liste des composants connexes du graphe G.

   ```
   ?- compconnexes([[2,7],[1],[4],[3,5],[4],[7],[1,6],[]],L).
   L = [[8],[6,1,2,7],[3,4,5]]
   ```

7. Ecrire le prédicat `cc(+G)`, qui affiche les composants connexes du graphe G.

   ```
   ?- cc([[2,7],[1],[4],[3,5],[4],[7],[1,6],[]]).
   Les composants connexes du graphe sont :
   [8]
   [6,1,2,7]
   [3,4,5]
   ```