DEUG MIAS 2e année
TD de programmation fonctionnelle n° 5
correction
Les graphes

Pierre Boulet   Jean-Marc Nicod   Frédéric Vivien

1 Chemin entre deux sommets d'un arbre

On se donne un arbre sous la forme d'une fonction successeur: *succ*. Cette fonction prend en argument la clef (le nombre) associé à un sommet et renvoie la liste des clefs de ses fils. Cette fonction renvoie la liste vide pour un sommet qui n'a pas de fils.

Question 1.1. Écrivez la fonction *succ* pour l'arbre de la figure 1.

```
(define (succ n)
    (cond ((= n 1) '(2 3 5))
          ((= n 3) '(6 9 15))
          ((= n 5) '(10 20))
          ((= n 9) '(4 7 12))
          (else '())
    )
)
```

Un chemin dans l'arbre est représenté par la suite des sommets traversés.

Question 1.2. Écrivez la fonction *prolonge* qui prend en argument un chemin c représenté par une liste et qui renvoie la liste de tous les chemins obtenus en prolongeant c d'une arête. On supposera que le premier sommet de la liste représentant c est le dernier sommet visité, autrement dit le chemin qui utilise l'arête $1 \rightarrow 3$ est codé (3 1).

```
(define (prolonge l)
    (if (null? (succ (car l))) '()
       (map (lambda (x)(cons x l)) (succ (car l)))))
```

Question 1.3. Écrivez une fonction *chemin* qui prend en entrée un sommet *source* et un sommet *but* et qui renvoie le chemin de l'arbre de *source* vers *but* s'il existe, et qui renvoie la liste vide sinon. Écrivez deux versions de cette fonction : une qui réalise un parcours en profondeur et l'autre en largeur.

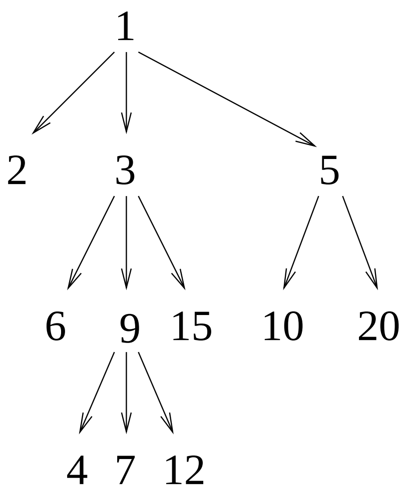

FIG. 1 - *Exemple d'arbre*

```
(define (chemin-prof source but)
    (define (partde l)
       (let ((courant (if (null? l) () (car l))))
          (if (null? l) #f
             (cond ((equal? courant ()) (partde (cdr l)))
                   ((= (car courant) but) courant)
                   (else (partde (append (prolonge courant) (cdr l))))))))
    (partde (list (list source))))

(define (chemin-larg source but)
    (define (partde l)
       (let ((courant (if (null? l) () (car l))))
          (if (null? l) #f
             (cond ((equal? courant ()) (partde (cdr l)))
                   ((= (car courant) but) courant)
                   (else (partde (append (cdr l) (prolonge courant))))))))
    (partde (list (list source))))
```

2 Plus court chemin dans un graphe sans circuits

Nous considérons maintenant des graphes valués.

Question 2.1. Écrivez la fonction *succ* pour le graphe de la figure 2. Ne pas oublier d'associer à chaque successeur la distance qui le sépare de son prédecesseur.

```
(define (succ x)
    (cond ((eq? x 's) '((a 6)(b 4)))
          ((eq? x 'a) '((c 8)(d 10)))
          ((eq? x 'b) '((f 20)))
```

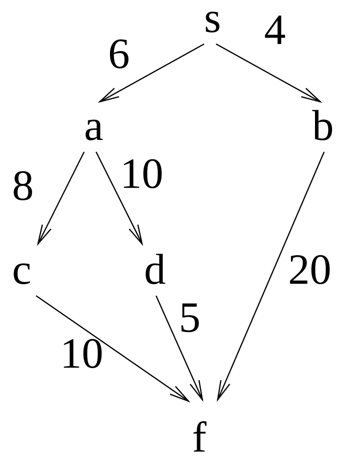

FIG. 2 - *Exemple de graphe valué*

```
        ((eq? x 'c) '((f 10)))
        ((eq? x 'd) '((f 5)))
        (else '())))
```

On représente un chemin par une liste. On codera le chemin de s à b de la forme: $((b\ 10)(s\ 0))$, c'est-à-dire en associant à tout sommet du chemin la distance qui le sépare de son prédecesseur.

Question 2.2. Écrivez la fonction *prolonge* qui prend en argument un chemin c représenté par une liste et qui renvoie la liste de tous les chemins obtenus en prolongeant c d'une arête.

```
(define (prolonge l)
   (if (null? (succ (caar l)))
       ()
       (map (lambda (x)(cons x l)) (succ (caar l)))))
```

Question 2.3. Écrivez la fonction *longueur* qui prend en argument un chemin c et qui renvoie sa longueur.

```
(define (longueur chemin)
   (if (null? chemin)
      0
      (apply + (map (lambda (x) (if (null? x) 0 (cadr x))) chemin))))
```

Question 2.4. Écrivez la fonction *plus-court* qui prend en argument une liste de chemins et qui renvoie le chemin le plus court.

```
(define (plus-court l)
   (define (moins-long l min)
      (if (null? l)
         min
```

```
        (if (> (longueur min) (longueur (car l)))
               (moins-long (cdr l) (car l))
               (moins-long (cdr l) min))))
  (if (null? l)
      ()
      (moins-long (cdr l) (car l))))
```

Question 2.5. Écrivez une fonction *chemin* qui prend en entrée un sommet *source* et un sommet *but* et qui renvoie le chemin de l'arbre de *source* vers *but* s'il existe et sa longueur, et qui renvoie la liste vide sinon.

```
(define (meilleur source but)
  (define (partde l)
    (letrec ((courant (plus-court l))
             (enleve (lambda (x l)
                       (cond ((null? l) ())
                             ((equal? x (car l)) (cdr l))
                             (else (cons (car l) (enleve x (cdr l))))))))
      (if (null? l)
          #f
          (if (eq? (caar courant) but) courant
              (partde (append (enleve courant l) (prolonge
              courant)))))))
  (partde (list (list (list source 0)))))
```

3 Graphe donné en extension: tri topologique

Un graphe est maintenant défini par une liste de listes composées d'un sommet et de ses successeurs. Exemple :

```
(define Graphe '((a b f) (b) (c d e) (d a e) (e) (f)))
```

Question 3.1. Écrivez la fonction *sommets* qui génère la liste des sommets d'un graphe.

```
(define (sommets graphe)
  (map car graphe))
```

Question 3.2. Écrivez la fonction *succ* pour cette représentation.

```
(define (succ sommet graphe)
  (if (null? graphe)
      ()
      (if (eq? sommet (caar graphe))
          (cdr (car graphe))
          (succ sommet (cdr graphe)))))
```

Question 3.3. Écrivez la fonction *supprimer* qui supprime un sommet d'un graphe.

```
(define (supprimer s g)
  (let ((enlever (lambda (x l)
                   (if (null? l)
                       ()
                       (if (equal? x (car l))
                           (cdr l)
                           (cons (car l) (enlever x (cdr l))))))))
    (if (null? g)
        ()
        (if (equal? s (caar g))
            (supprimer s (cdr g))
            (cons (cons (caar g) (enlever s (cdar g))) (supprimer s (cdr g)))))))
```

Question 3.4. Écrivez la fonction *sans-successeurs* qui renvoie la liste des sommets du graphe qui sont sans successeurs.

```
(define (sans-successeurs g)
    (if (null? g)
        ()
        (if (= 1 (length (car g)))
            (cons (caar g) (sans-successeurs (cdr g)))
            (sans-successeurs (cdr g)))))
```

Question 3.5. Écrivez la fonction *tri-topologique* qui renvoie la liste des sommets du graphe triés dans l'ordre topologique. Un sommet qui n'a pas de successeurs est dernier dans l'ordre topologique.

```
(define (tri-topologique g)
    (letrec ((supprimer-l (lambda (l g)
                            (if (null? l)
                                g
                                (supprimer-l (cdr l) (supprimer (car l) g))))))
    (if (null? g)
        ()
        (append (tri-topologique (supprimer-l (sans-successeurs g) g)) (sans-successeurs g)))))
```