

Programmation fonctionnelle

Semestre 6

Professeur : Monsieur PIECHOWIAK Sylvain

l) Partie 1

a) L'énoncé

PARTIE 1: Calculs d'intégrales ...



On souhaite calculer une valeur approchée de l'intégrale d'une fonction réelle f supposée intégrable sur un intervalle $[a,b]$ en calculant l'aire algébrique de la surface limitée par l'axe des abscisses, la courbe qui représente f et les droites $y = a$ et $y = b$.

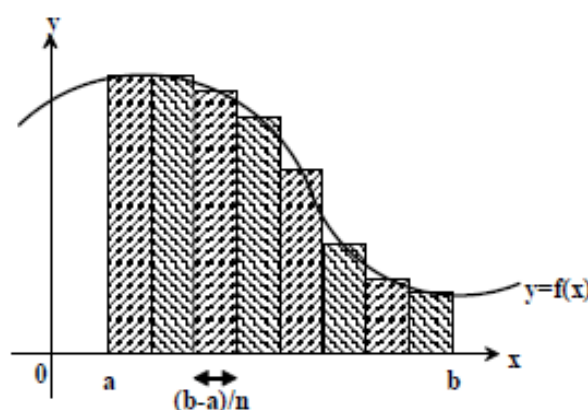
On découpe l'intervalle $[a,b]$ en n segments de même longueur $\frac{b-a}{n}$ et on calcule la somme des surface des rectangles de largeur $\frac{b-a}{n}$ et de hauteur la valeur de f pour le point du milieu de chaque segment.

L'abscisse du milieu du k ème segment vaut $m_k = a + (k-1)\frac{b-a}{n} + \frac{1}{2}\frac{b-a}{n}$. L'aire du k ème rectangle A_k vaut

donc $A_k = \frac{b-a}{n} \times f(m_k)$ et finalement pour un nombre n de rectangles la valeur approchée $\int_a^b f$ vaut donc :

$$\sum_{k=1}^n A_k = \frac{b-a}{n} \times \sum_{k=1}^n f(m_k)$$

Définir une fonction **integrale** qui pour le quadruplet (f, n, a, b) renvoie la valeur de l'intégrale $\int_a^b f$ obtenue par la méthode précédente avec n donné. f étant la fonction à intégrer, a et b étant les bornes de l'intégrale et n étant le pas de découpage. On supposera que la fonction f est continue sur $[a,b]$.



b) Le compte-rendu

On nous demande de calculer l'intégrale pour une fonction f dans un intervalle $[a,b]$ et n donnée. A la suite de l'énoncé, on sait que pour calculer l'intégrale, il sera nécessaire de calculer l'aire de l'ensemble des rectangles dans l'intervalle $[a,b]$ à l'aide la formule $A_k = \frac{b-a}{n} \times f(m_k)$ où m_k est l'abscisse du milieu du $k_{\text{ième}}$ segment, on peut le calculer à l'aide de la formule suivante : $m_k = a + (k-1) \frac{b-a}{n} + \frac{1}{2} \frac{b-a}{n}$. Donc pour résumer, afin de calculer l'intégrale, on va commencer par calculer le milieu m_k et l'aire du rectangle pour ensuite pouvoir calculer A_k puis on obtiendra l'intégrale en faisant la somme des aires de l'ensemble des rectangles contenu dans l'intervalle.

1. Calcul de m_k

```
(define milieu
  (lambda (k a b n)
    (+ a (* (- k 1) (/ (- b a) n)) (/ (- b a) (* 2 n)))))
```

FIGURE 1 : SCRIPT PERMETTANT LE CALCUL DE L'ABSCISSE DU MILIEU DU SEGMENT K

2. Calcul de l'aire d'un rectangle

```
(define aireRectangle
  (lambda (L l)
    (* L l)))
```

FIGURE 2 : SCRIPT PERMETTANT LE CALCUL DE L'AIRES D'UN RECTANGLE

3. Calcul de A_k

```
(define calculIntegrale
  (lambda (f n a b i)
    (if (= i 0)
        0
        (+ (aireRectangle (/ (- b a) n) (f (milieu i a b n))) (calculIntegrale f n a b (- i 1)))))
```

FIGURE 3 : SCRIPT PERMETTANT LE CALCUL A_k

4. Calcul de l'intégrale

```
(define integrale
  (lambda (f n a b)
    (calculIntegrale f n a b n)))
```

FIGURE 4 : SCRIPT PERMETTANT LE CALCUL DE L'INTEGRALE

Exemple :

1. Prenons l'exemple de la fonction $f(x) = x^3$. On va chercher l'intégrale de cette fonction dans l'intervalle $[2,4]$ et où $n=1$.

```
(integrale f 1 2 4)
```

FIGURE 5 : SCRIPT PERMETTANT L'APPEL A NOTRE FONCTION INTEGRALE

```
Welcome to DrRacket, version 8.0 [cs].  
Language: racket, with debugging; memory limit: 128 MB.  
54
```

FIGURE 6 : RESULTAT DE L'APPEL A LA FONCTION INTEGRALE

Notre calcul d'intégrale vaut 54.

II) Partie 2

a) L'énoncé

PARTIE 2 : Calculs de polynômes ...

On se propose de manipuler formellement des polynômes, c'est à dire travailler sur leur expression. On rappelle qu'un polynôme est une expression de la forme :

$$P(X) = a_n X^n + a_{n-1} X^{n-1} + \dots + a_1 X^1 + a_0 X^0 = \sum_{i=0}^n a_i X^i$$

où n est appelé le degré du polynôme et où chaque $a_i X^i$ est appelé monôme de degré i et de coefficient a_i

Chaque monôme $a_i X^i$ sera représenté par un couple (a_i, i) et chaque polynôme sera représenté par la liste de ses monômes $((a_0, 0), (a_1, 1), (a_2, 2), \dots, (a_n, n))$. On notera que les monômes ne sont pas nécessairement rangés selon l'ordre des degrés.

- 1) Comment sera représenté le polynôme : $P(X) = 7X^9 - 3X^7 + 2X^2 + 5$?
- 2) Donner une fonction **degreMonome** qui renvoie le degré du monôme qu'on lui passe en argument
- 3) Donner une fonction **coefficientMonome** qui renvoie le coefficient du monôme qu'on lui passe en argument
- 4) Donner une fonction **degrePolynome** qui renvoie le degré du polynôme qu'on lui passe en argument.
- 5) Donner une fonction qui renvoie la valeur d'un monôme pour une valeur de X précise. On appellera **valeurMonome** cette fonction. EX : (**valeurMonome** ' (2 3) 2) renvoie 16.
- 6) Donner une fonction qui renvoie la valeur d'un polynôme pour une valeur de X précise. On appellera **valeurPolynome** cette fonction.
- 7) Donner une fonction qui renvoie la somme formelle de 2 polynômes.
- 8) Donner une fonction qui renvoie la dérivée formelle d'un polynôme. On rappelle que la dérivée du polynôme $P(X)$ est $P'(X)$ avec :

$$P'(X) = n \times a_n X^{n-1} + (n-1) \times a_{n-1} X^{n-2} + \dots + 1 \times a_1 X^0$$

- 9) Donner une fonction qui renvoie la primitive formelle d'un polynôme.
- 10) Donner une fonction qui renvoie le produit formel de 2 polynômes.

b) Le compte-rendu

- 1) Un polynôme est une expression de la forme suivante :

$$P(X) = a_n X^n + a_{n-1} X^{n-1} + \dots + a_1 X^1 + a_0 X^0 = \sum_{i=0}^n a_i X^i$$

Nous allons donc représenter chaque monôme par un couple (a_i, i) où a_i est la valeur et i est le degré. Le polynôme sera donc représenté par une liste de monôme. Pour le polynôme $7X^9 - 3X^7 + 2X^2 + 5$, on aura donc $((5\ 0)\ (2\ 2)\ (-3\ 7)\ (7\ 9))$

- 2) On veut renvoyer le degré d'un monôme. Pour rappel, un monôme est de la forme (a_i, i) où a_i est la valeur et i est le degré. Ainsi pour renvoyer le degré, il va falloir récupérer i qui est la tête de la queue du monôme.

```
(define degreMonome  
  (lambda (M)  
    (car (cdr M))))
```

FIGURE 7 : SCRIPT PERMETTANT DE RECUPERER LE DEGRE DU MONOME

- 3) On veut renvoyer le coefficient d'un monôme. Pour renvoyer le coefficient, il va falloir récupérer a_i qui est la tête du monôme.

```
(define coefficientMonome  
  (lambda (M)  
    (car M)))
```

FIGURE 8 : SCRIPT PERMETTANT DE RECUPERER LE COEFFICIENT DU MONOME

- 4) On veut renvoyer le degré d'un polynôme.
- Dans le cas où le polynôme est nul, le degré maximal est 0.
 - Sinon il faudra vérifier si le degré de la tête du polynôme est supérieur à la valeur que l'on « tient en mémoire ».
 - Dans le cas où c'est vrai, on va appeler la fonction avec la liste sans l'élément de tête du polynôme et prendre une nouvelle valeur en mémoire.
 - Dans le cas où c'est faux, on va appeler la fonction en appelant la liste sans l'élément de tête et en laissant la même valeur en mémoire que lors du premier appel.

```
(define degrePolynome
  (lambda (P)
    (degrePolynome2 P 0)))

(define degrePolynome2
  (lambda (P D)
    (if (null? P)
        D
        (if (> (degreMonome (car P)) D)
            (degrePolynome2 (cdr P) (degreMonome (car P)))
            (degrePolynome2 (cdr P) D)))))
```

FIGURE 9 : SCRIPT PERMETTANT DE RECUPERER LE DEGRE MAXIMAL DU POLYNOME

- 5) On veut renvoyer la valeur d'un monôme avec X donné. Il faudra donc calculer X à la puissance de l'indice du monôme. Le tout multiplier par le coefficient du monôme.

```
(define valeurMonome
  (lambda (M X)
    (* (expt X (degreMonome M)) (coefficientMonome M))))
```

FIGURE 10 : SCRIPT PERMETTANT DE RECUPERER LA VALEUR DU MONOME

- 6) On veut renvoyer la valeur d'un polynôme avec X donné. Il faudra donc calculer la valeur de chaque monôme et en faire la somme. On va donc calculer dans un premier temps la valeur du monôme de tête et appeler la fonction avec le polynôme sans le premier élément et ainsi de suite en faisant la somme des résultats.

```
(define valeurPolynome
  (lambda (P X)
    (valeurPolynome2 P X 0)))

(define valeurPolynome2
  (lambda (P X R)
    (if (null? P)
        R
        (+ (valeurMonome (car P) X) (valeurPolynome2 (cdr P) X R)))))
```

FIGURE 11 : SCRIPT PERMETTANT DE RECUPERER LA VALEUR DU POLYNOME

- 7) On veut pouvoir faire la somme formelle de deux polynômes. Il faudra donc trouver les monômes qui ont le même degré et faire la somme des coefficients. Une fois la somme faites, on va créer une nouvelle liste qui va contenir les monômes additionnés.

```
(define sommeFormelle
  (lambda (P1 P2)
    (if (null? P1)
        P2
        (sommeFormelle2 P1 P2 '()))
  )
)

(define sommeFormelle2
  (lambda (P1 P2 Ps)
    (if (> (degrePolynome P1) (degrePolynome P2))
        (sommeFormelle3 P1 P2 0 (degrePolynome P2) '())
        (sommeFormelle3 P1 P2 0 (degrePolynome P2) '())
    )
  )
)

(define sommeFormelle3
  (lambda (P1 P2 i nbMax PS)
    (if (= i (+ nbMax 1))
        PS
        (if (= 0 (coefficientMonome (sommeMonome (iemeMonome P1 i) (iemeMonome P2 i))))
            (sommeFormelle3 P1 P2 (+ i 1) nbMax PS)
            (sommeFormelle3 P1 P2 (+ i 1) nbMax (append PS (list (sommeMonome (iemeMonome P1 i) (iemeMonome P2 i))))))
    )
  )
)
)
```

FIGURE 12 : SCRIPT PERMETTANT LE CALCUL DE L'AIRES D'UN RECTANGLE

```
(define sommeMonome
  (lambda (M1 M2)
    (list (+ (car M1) (car M2)) (cdr M1))
  )
)

(define iemeMonome
  (lambda (Po i)
    (if (null? Po)
        (list 0 i)
        (if (= (degreMonome (car Po)) i)
            (car Po)
            (iemeMonome (cdr Po) i)
        )
    )
  )
)
)
```

FIGURE 13 : SCRIPT PERMETTANT LE CALCUL DE L'AIRES D'UN RECTANGLE

- 8) On veut renvoyer la dérivée formelle d'un polynôme. Pour rappel, la dérivée formelle d'un polynôme est de la forme :

$$P'(X) = n \times a_n X^{n-1} + (n-1) \times a_{n-1} X^{n-2} + \dots + 1 \times a_1 X^0$$

Il va falloir créer une nouvelle liste en ajoutant le monôme dérivé. Pour dériver un monôme, il faudra vérifier que le degré est supérieur à 1, sinon pour trouver le nouveau coefficient, il suffit de multiplier le coefficient par le degré puis pour trouver le nouveau degré, il faut enlever 1 au degré initial.

```
(define derive
  (lambda (P)
    (deriveFormelle P '())))

(define deriveFormelle
  (lambda (P1 P2)
    (if (null? P1)
        P2
        (if (> (degreMonome (car P1)) 1)
            (deriveFormelle (cdr P1) (append P2 (list(list (* (degreMonome (car P1)) (coefficientMonome (car P1)) ) (- (degreMonome (car P1)) 1) ) )))
            (deriveFormelle (cdr P1) P2)
          )
        )
    )
  )
)
```

FIGURE 14 : SCRIPT PERMETTANT LE CALCUL DE L'AIRES D'UN RECTANGLE

- 9) On veut renvoyer la primitive formelle d'un polynôme. Pour rappel, la primitive formelle d'un polynôme est de la forme :

$$P(X) = \frac{a_n}{n+1} X^{n+1} + \dots$$

Il va falloir créer une nouvelle liste en ajoutant la primitive du monôme. Pour trouver la primitive du monôme, il faudra vérifier que le degré n'est pas nul, sinon pour trouver le nouveau coefficient, il suffit de multiplier le coefficient par $\frac{1}{\text{degré}+1}$ puis pour trouver le nouveau degré, il faut ajouter 1 au degré initial.

```
(define primitive
  (lambda (P)
    (primitiveFormelle P '())))

(define primitiveFormelle
  (lambda (P1 P2)
    (if (null? P1)
        P2
        (if (= (degreMonome (car P1)) 0)
            (primitiveFormelle (cdr P1) (append P2 (list(list (/ (coefficientMonome (car P1)) (+ 1 (degreMonome (car P1)))) (+ (degreMonome (car P1)) 1) ) )))
            (primitiveFormelle (cdr P1) (append P2 (list(list (/ (coefficientMonome (car P1)) (+ 1 (degreMonome (car P1)))) (+ (degreMonome (car P1)) 1) ) )))
          )
        )
    )
  )
)
```

FIGURE 15 : SCRIPT PERMETTANT LE CALCUL DE L'AIRES D'UN RECTANGLE

- 10) On veut pouvoir faire la multiplication formelle de deux polynômes. On va prendre le premier monôme du polynôme 1 puis on va le multiplier au second polynôme. Pour multiplier le monôme et le polynôme, on va multiplier les coefficients et faire la somme des degrés pour obtenir le produit. Et ainsi de suite, on prendra toujours un monôme à la fois du polynôme 1. Le produit des deux polynômes sera contenu dans une nouvelle liste. Une fois que l'on a obtenu le produit des 2 polynômes, on va pouvoir faire la somme du résultat obtenu.

```
(define multiplication
  (lambda (P1 P2)
    (multiplicationFormelle P1 P2 '())))

(define multiplicationFormelle
  (lambda (P1 P2 Ps)
    (if (null? P1)
        Ps
        (multiplicationFormelle (cdr P1) P2 (sommeFormelle Ps (multMoParPo (car P1) P2 '())))))
  )

(define multMoParPo
  (lambda (Mo P1 P2)
    (if (null? P1)
        P2
        (multMoParPo Mo (cdr P1) (append P2 (list (list (* (coefficientMonome Mo) (coefficientMonome (car P1)) ) (+ (degreMonome (car P1)) (degreMonome Mo)) ) ))))
  )
  )
```

FIGURE 16 : SCRIPT PERMETTANT LE MULTIPLICATION DE 2 POLYNOMES

Résultats :

On va tester l'ensemble des fonctions préalablement créés avec les polynômes

$$P(x) = 7x^9 - 3x^7 + 2x^2 + 5$$

```
(define P '((5 0) (2 2) (-3 7) (7 9))
```

$$P1(x) = 4x^8 - 3x^4 + 4x^2 + 15$$

```
(define P1 '((15 0) (4 2) (3 4) (4 8))
```

Test des fonctions respectivement degreMonome, coefficientMonome, degrePolynome, valeurMonome, valeurPolynome, derive et primitive avec le polynôme P(x) et le monôme $5X^4$ et avec X=2 pour les fonctions valeurMonome et valeurPolynome.

```
Welcome to DrRacket, version 8.0 [cs].
Language: racket, with debugging; memory limit: 128 MB.
4
5
9
80
3213
'((4 1) (-21 6) (63 8))
'((5 1) (2/3 3) (-3/8 8) (-7/10 10))
```

FIGURE 17 : RESULTATS DES APPELS AUX FONCTIONS

- 4 est bien le degré du monome $5X^4$
- 5 est bien le coefficient du monome $5X^4$
- 9 est bien le degre maximal du polynome P(x)
- 16 est bien le résultat du monome en remplaçant x par 2

$$= 5 * 2^4 = 80$$
- 3213 est bien le résultat du polynome en remplaçant x par 2.

$$= 7 * 2^9 - 3 * 2^7 + 2 * 2^2 + 5 = 3213$$

- '((4 1) (-21 6) (63 8)), résultat à lire : $= 63X^8 - 21X^6 + 4X$. C'est bien le dérivé de $P(x)$
- '((5 1) (2/3 3) (-3/8 8) (7/10 10)), résultat à lire : $= \frac{7}{10}X^{10} - \frac{3}{8}X^8 + \frac{2}{3}X^3 + 5X$. C'est bien le primitive de $P(x)$

Test des fonctions respectivement somme formelle et multiplication avec les polynômes $P(x)$ et $P2(x)$.

```

Welcome to DrRacket, version 8.0 [cs].
Language: racket, with debugging; memory limit: 128 MB.
'((20 0) (6 2) (3 4) (-3 7) (4 8))
'((75 0) (50 2) (23 4) (6 6) (-45 7) (20 8) (93 9) (8 10) (19 11) (21 13) (-12 15) (28 17))

```

FIGURE 18 : RESULTATS DES APPELS AUX FONCTIONS

- '((20 0) (6 2) (3 4) (-3 7) (4 8)), résultat à lire : $= 4X^8 - 3X^7 + 3X^4 + 6X^2 + 20$
- '((75 0) (50 2) (23 4) (6 6) (-45 7) (20 8) (93 9) (8 10) (19 11) (21 13) (-12 15) (28 17)),
résultat à lire : $= 28X^{17} - 12X^{15} + 21X^{13} + 19X^{11} + 8X^{10} + 93X^9 + 20X^8 - 45X^7 + 6X^6 + 23X^4 + 50X^2 + 75$

III) Partie 3

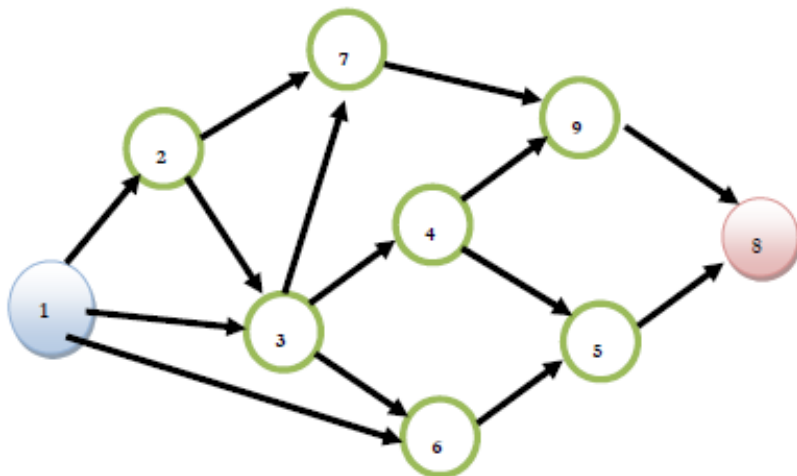
a) L'énoncé

PARTIE 3 : Sur les graphes ...

On souhaite manipuler des graphes qui représentent des plannings de construction d'une maison. Chaque nœud représente une étape dans la construction. Il est associé à une durée qui représente la durée de l'étape (par exemple en nombre de jours). Chaque arc indique une précédence entre 2 étapes. Dans l'exemple ci-dessous, l'étape 2 doit être réalisée avant les étapes 3 et 7. L'étape 1 est la première étape et 8 la dernière. On va utiliser ce graphe pour

suivre l'évolution de l'ensemble des travaux. On supposera qu'il n'y a pas de boucle dans le graphe.

Connaissant la date de début des travaux (date de l'étape 1) on peut déterminer la date au plus tôt de la fin du chantier. En suivant le sens des arcs et en utilisant la durée de chaque étape. De même, en définissant la date de fin des travaux, on peut déterminer la date au plus tard du début du chantier. En suivant l'ordre inverse des arcs.



- 1) Proposer une structure de donnée pour représenter un graphe.
- 2) Proposer les fonctions de base de manipulation d'un graphe :
 - a. Création d'un nœud
 - b. Ajout d'un nœud dans un graphe
 - c. Retrait d'un nœud dans un graphe
 - d. Liste des nœuds successeurs directs d'un nœud
 - e. Liste des prédécesseurs directs d'un nœud

On supposera que chaque graphe possède 2 nœuds particuliers appelés Debut et Fin et qui représentent la première et la dernière étape du chantier.

- 3) Connaissant la date au plus tôt de tous les nœuds prédécesseur d'un nœud donné, comment calculer la date au plus tôt de ce nœud ? Proposer une fonction qui détermine la date au plus tôt d'un nœud quelconque du graphe étant donnée la date de début souhaitée du chantier.
- 4) Connaissant la date au plus tard de tous les nœuds successeurs d'un nœud donné, comment se calcule la date au plus tard de ce nœud ? Proposer une fonction qui détermine la date au plus tard d'un nœud quelconque du graphe étant donnée la date de fin souhaitée du chantier.

Tant que la durée d'une étape reste comprise entre sa date au plus tôt et sa date au plus tard, on est certain que cette étape sera réalisée dans les temps souhaités. Sinon, on sait qu'un retard va se produire sur le chantier.

La criticité d'une étape est la différence entre la durée d'une étape et sa durée possible. Plus cette différence est petite plus l'étape est critique.

- 5) Proposer une fonction qui détermine la criticité d'une étape.
- 6) Proposer une fonction qui classe les étapes selon l'ordre de criticité. Les étapes les plus critiques étant situées au début du classement.

b) Le compte-rendu

- 1) On se dit de représenter un graphe. Pour cet exercice, nous le représenterons par une liste (n v LP LS) où n et v sont respectivement le nom et la valeur du nœud et LP et LS sont respectivement la liste des prédécesseurs et des successeurs du nœud.
- 2) On veut créer les fonctions de base dans la manipulation d'un graphe.
 - Création d'un nœud. On va donc devoir créer une liste qui contient le nom, la valeur, la liste des prédécesseurs et la liste des successeurs du nœud.

```
(define creationNoeud
  (lambda (n v LP LS)
    (list n v LP LS)))
```

FIGURE 19 : SCRIPT PERMETTANT LA CREATION D'UN NOEUD

- Pour m'aider dans la réalisation des fonctions suivantes, je vais créer les fonctions permettant de :
 - Récupérer la valeur du nœud. On va donc vouloir récupérer la tête de la queue de la liste.

```
(define valeurNoeud
  (lambda (N)
    (cadr N)))
```

FIGURE 20 : SCRIPT PERMETTANT DE RECUPERER LA VALEUR DU NOEUD

- Récupérer la liste des prédécesseurs. On va donc vouloir récupérer la tête de la queue de la queue de la liste.

```
(define noeudPred;
  (lambda (N)
    (caddr N)))
```

FIGURE 21 : SCRIPT PERMETTANT DE RECUPERER LA LISTE DES PREDECESSEURS

- Récupérer la liste des successeurs. On va donc vouloir récupérer la tête de la queue de la queue de la queue de la liste.

```
(define noeudSuc
  (lambda (N)
    (cadddr N)))
```

FIGURE 22 : SCRIPT PERMETTANT DE RECUPERER LA LISTE DES SUCCESSEURS

- Vérifier si une valeur appartient à une liste. On va donc vérifier si la liste est nulle, dans ce cas la valeur n'appartient pas à la liste. Dans le cas contraire, on va vérifier si la valeur de la tête de la liste est égal à la valeur. Si c'est vrai, on va retourner vrai sinon on va rappeler la fonction avec la liste sans l'élément de tête.

```
(define appartient?
  (lambda (e L)
    (if (null? L)
        #f
        (if (= (car L) e)
            #t
            (appartient? e (cdr L))))))
```

FIGURE 23 : SCRIPT PERMETTANT DE VERIFIER SI UN NOEUF APPARTIENT A UN GRAPHE

- Ajout d'un nœud dans un graphe.

On va d'abord vérifier que le graphe n'est pas nul. Dans le cas où il est nul, notre graphe ne sera composé que du nœud à ajouter. Dans le cas contraire, on devra :

- Ajouter notre nœud au graphe
- Ajouter le nom de notre nœud à la liste des successeurs de ses prédécesseurs
- Ajouter le nom de notre nœud à la liste des prédécesseurs de ses successeurs.

```
(define ajoutNoeud
  (lambda (N G)
    (if (null? G)
        N
        (if (null? N)
            G
            (ajoutNoeud2 G N '())))))

(define ajoutNoeud2
  (lambda (G N G2)
    (if (null? G)
        (cons N G2)
        (if (appartient? (nomNoeud (car G)) (noeudSuc N))
            (ajoutNoeud2 (cdr G) N (cons (creationNoeud (nomNoeud (car G)) (valeurNoeud (car G)) (cons (nomNoeud N) (noeudPred (car G)) (noeudSuc (car G))) G2))
            (if (appartient? (nomNoeud (car G)) (noeudPred N))
                (ajoutNoeud2 (cdr G) N (cons (creationNoeud (nomNoeud (car G)) (valeurNoeud (car G)) (noeudPred (car G)) (cons (nomNoeud N) (noeudSuc (car G))) G2))
                (ajoutNoeud2 (cdr G) N (cons (car G) G2)))))))
```

FIGURE 24 : SCRIPT PERMETTANT D'AJOUTER UN NOEUD DANS UN GRAPHE

- Retrait d'un nœud dans un graphe.

On va d'abord vérifier que le graphe n'est pas nul. Dans le cas où il est nul, notre graphe sera une liste vide. Dans le cas contraire, on va vérifier que le nœud n'est pas vide, dans le cas contraire on retournera le graphe sinon il faudra devra :

- Supprimer notre nœud au graphe
- Supprimer le nom de notre nœud dans la liste des successeurs de ses prédécesseurs
- Supprimer le nom de notre nœud à la liste des prédécesseurs de ses successeurs.

```
(define suppressionNoeud
  (lambda (N G)
    (if (null? G)
        '()
        (if (null? N)
            G
            (suppressionNoeud2 G N '())))))

(define suppressionNoeud2
  (lambda (G N G2)
    (if (null? G)
        (cons G2 '())
        (if (appartient? (nomNoeud N) (noeudSuc (car G)))
            (suppressionNoeud2 (cdr G) N G2)
            (if (appartient? (nomNoeud N) (noeudPred (car G)))
                (suppressionNoeud2 (cdr G) N)
                (suppressionNoeud2 (cdr G) N (cons (car G) G2)))))))
```

FIGURE 25 : SCRIPT PERMETTANT DE SUPPRIMER UN NOEUD DANS UN GRAPHE

- Liste des nœuds successeurs directs d'un nœud. Pour trouver la liste des successeurs directs d'un noeud N dans un graphe G, on va chercher si le noeud de nom N se trouve dans le graphe G, si c'est le cas on retourne sa liste des successeurs, sinon on retourne une liste vide.

```
(define listeSuccesseurs
  (lambda (N G)
    (if (null? G)
        '()
        (if (eq? N (nomNoeud (car G)))
            (noeudSuc (car G))
            (listeSuccesseurs N (cdr G)))))
```

FIGURE 26 : SCRIPT PERMETTANT DE RECUPERER LA LISTE DES SUCCESSEURS DIRECTS D'UN NOEUD

- Liste des prédécesseurs directs d'un nœud. Pour trouver la liste des prédécesseurs directs d'un noeud N dans un graphe G, on va chercher si le noeud de nom N se trouve dans le graphe G, si c'est le cas on retourne sa liste de prédécesseurs, sinon on retourne une liste vide.

```
(define listePredecesseurs
  (lambda (N G)
    (if (null? G)
        '()
        (if (eq? N (nomNoeud (car G)))
            (noeudPred (car G))
            (listePredecesseurs N (cdr G)))))
```

FIGURE 27 : SCRIPT PERMETTANT DE RECUPERER LA LISTE DES PREDECESEURS DIRECTS D'UN NOEUD

- 3)
- 4)
- 5) On veut calculer la criticité d'une étape. Pour rappel, la criticité d'une étape est la différence entre la durée d'une étape et sa durée possible. Il va donc falloir récupérer la valeur de la durée d'une étape et la valeur de sa durée possible. L'ensemble de ses valeurs se trouvent dans la valeur du nœud. La durée d'une étape sera la tête de la valeur du nœud et la durée possible est la tête de la queue de la valeur du nœud.

```
(define criticiteEtape
  (lambda (N)
    (if (null? N)
        0
        (- (car(valeurNoeud N)) (car(cdr(valeurNoeud N))))))
```

FIGURE 28 : SCRIPT PERMETTANT LE CALCUL DE LA CRITICITE D'UNE ETAPE

6)

Résultats :

On va tester l'ensemble des fonctions préalablement créés avec le graphe qui se trouve dans l'énoncé.

```
(define G (list '(1 2 ()) (2 3 6)) '(2 2 (1) (3 7)) '(3 2 (1 2) (4 6 7)) '(4 2 (3) (5 9)) '(5 2 (4 6) (8)) '(6 2 (1 3) (5)) '(7 2 (2 3) (9)) '(8 2 (5 9) ()))
```

FIGURE 6 : SCRIPT DU GRAPHE G

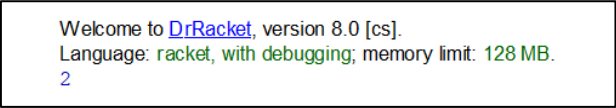
Test des fonctions respectivement creationNoeud, ajoutNoeud, suppressionNoeud, listePredecesseurs, listeSuccesseurs, avec le graphe G et la creation d'un nœud N =(10 '(4 2) '(1) '(3 4)).

```
Welcome to DrRacket, version 8.0 [cs].
Language: racket, with debugging, memory limit: 128 MB.
* ((10 (4 2) (1) (3 4)) (9 2 (4 7) (8)) (8 2 (5 9) (1)) (7 2 (2 3) (9)) (6 2 (1 3) (5)) (5 2 (4 6) (8)) (4 2 (10 3) (5 9)) (3 2 (10 1 2) (4 6 7)) (2 2 (1) (3 7)) (1 2 (1) (10 2 3 6)))
* (((9 2 (4 7) (8)) (8 2 (5 9) (1)) (7 2 (2 3) (9)) (6 2 (1 3) (5)) (5 2 (4 6) (8)) (4 2 (3) (5 9)) (3 2 (1 2) (4 6 7)) (2 2 (1) (3 7)) (1 2 (1) (2 3 6))))
* (1 2)
* (4 6 7)
```

FIGURE 29 : RESULTATS DES APPELS AUX FONCTIONS

- '((10 (4 2) (1) (3 4)) (9 2 (4 7) (8)) (8 2 (5 9) (1)) (7 2 (2 3) (9)) (6 2 (1 3) (5)) (5 2 (4 6) (8)) (4 2 (10 3) (5 9)) (3 2 (10 1 2) (4 6 7)) (2 2 (1) (3 7)) (1 2 (1) (10 2 3 6))) : On observe bien que le nœud N a été ajouté au graphe G.
- '(((9 2 (4 7) (8)) (8 2 (5 9) (1)) (7 2 (2 3) (9)) (6 2 (1 3) (5)) (5 2 (4 6) (8)) (4 2 (3) (5 9)) (3 2 (1 2) (4 6 7)) (2 2 (1) (3 7)) (1 2 (1) (2 3 6)))) : On observe bien que nœud N a été supprimé du graphe G.
- '(1 2) est bien la liste des prédécesseurs du nœud ayant le nom 3.
- '(4 6 7) est bien la liste des successeurs du nœud ayant le nom 3.

Test des fonctions respectivement `criticiteEtape` avec le nœud $N = (10 \ '(4 \ 2) \ '(1) \ '(3 \ 4))$.



```
Welcome to DrRacket, version 8.0 [cs].  
Language: racket, with debugging; memory limit: 128 MB.  
2
```

FIGURE 6 : RESULTATS DES APPELS AUX FONCTIONS

- On observe bien que la différence entre la durée d'une étape et sa durée possible est 2.